NUMERICAL SIMULATION OF DISPERSE PARTICLE FLOWS ON A GRAPHICS PROCESSING UNIT

by

Adam J. Sierakowski

A dissertation submitted to the Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

March, 2016

Abstract

In both nature and technology, we commonly encounter solid particles being carried within fluid flows, from dust storms to sediment erosion and from food processing to energy generation. The motion of uncountably many particles in highly dynamic flow environments characterizes the tremendous complexity of such phenomena. While methods exist for the full-scale numerical simulation of such systems, current computational capabilities require the simplification of the numerical task with significant approximation using closure models widely recognized as insufficient. There is therefore a fundamental need for the investigation of the underlying physical processes governing these disperse particle flows.

In the present work, we develop a new tool based on the Physalis method for the first-principles numerical simulation of thousands of particles (a small fraction of an entire disperse particle flow system) in order to assist in the search for new reduced-order closure models. We discuss numerous enhancements to the efficiency and stability of the Physalis method, which introduces the influence of spherical particles to a fixed-grid incompressible Navier-Stokes flow solver using a local analytic

solution to the flow equations.

Our first-principles investigation demands the modeling of unresolved length and time scales associated with particle collisions. We introduce a collision model alongside Physalis, incorporating lubrication effects and proposing a new nonlinearly damped Hertzian contact model. By reproducing experimental studies from the literature, we document extensive validation of the methods.

We discuss the implementation of our methods for massively parallel computation using a graphics processing unit (GPU). We combine Eulerian grid-based algorithms with Lagrangian particle-based algorithms to achieve computational throughput up to 90 times faster than the legacy implementation of Physalis for a single central processing unit. By avoiding all data communication between the GPU and the host system during the simulation, we utilize with great efficacy the GPU hardware with which many high performance computing systems are currently equipped. We conclude by looking forward to the future of Physalis with multi-GPU parallelization in order to perform resolved disperse flow simulations of more than 100,000 particles and further advance the development of reduced-order closure models.

Advisor: Andrea Prosperetti

Primary Reader: Rajat Mittal

Secondary Reader: Robert A. Dalrymple

Preface

The vast majority of the work presented in this dissertation grew rather organically from the first project that I was assigned to work on by my doctoral advisor, Andrea Prosperetti, at the Johns Hopkins University in 2011. The goal of the project was to perform a parametric study of the harmonic oscillation of a sphere in a viscous fluid, and I inherited a code that implemented the Physalis method—a method of utmost importance to this work that we will discuss at length in due time—with which to perform the simulations. The simulation, pictured in Figure 0.1, comprised of a sphere attached to the center of a periodic domain by a linear spring that was displaced from the center and released from rest. Using a modest domain of resolution $64 \times 32 \times 32$, the first of more than twenty simulations in which we would vary the ratio of the particle density to the fluid density, the fluid viscosity, and the spring stiffness, required nearly three weeks to run.

Wholly unsatisfied with the prospect of waiting a year to obtain a single (tiny) dataset, Professor Prosperetti and I decided that it would be worth the risk to rewrite the code for parallel processing in the hope of making it faster. With some experience

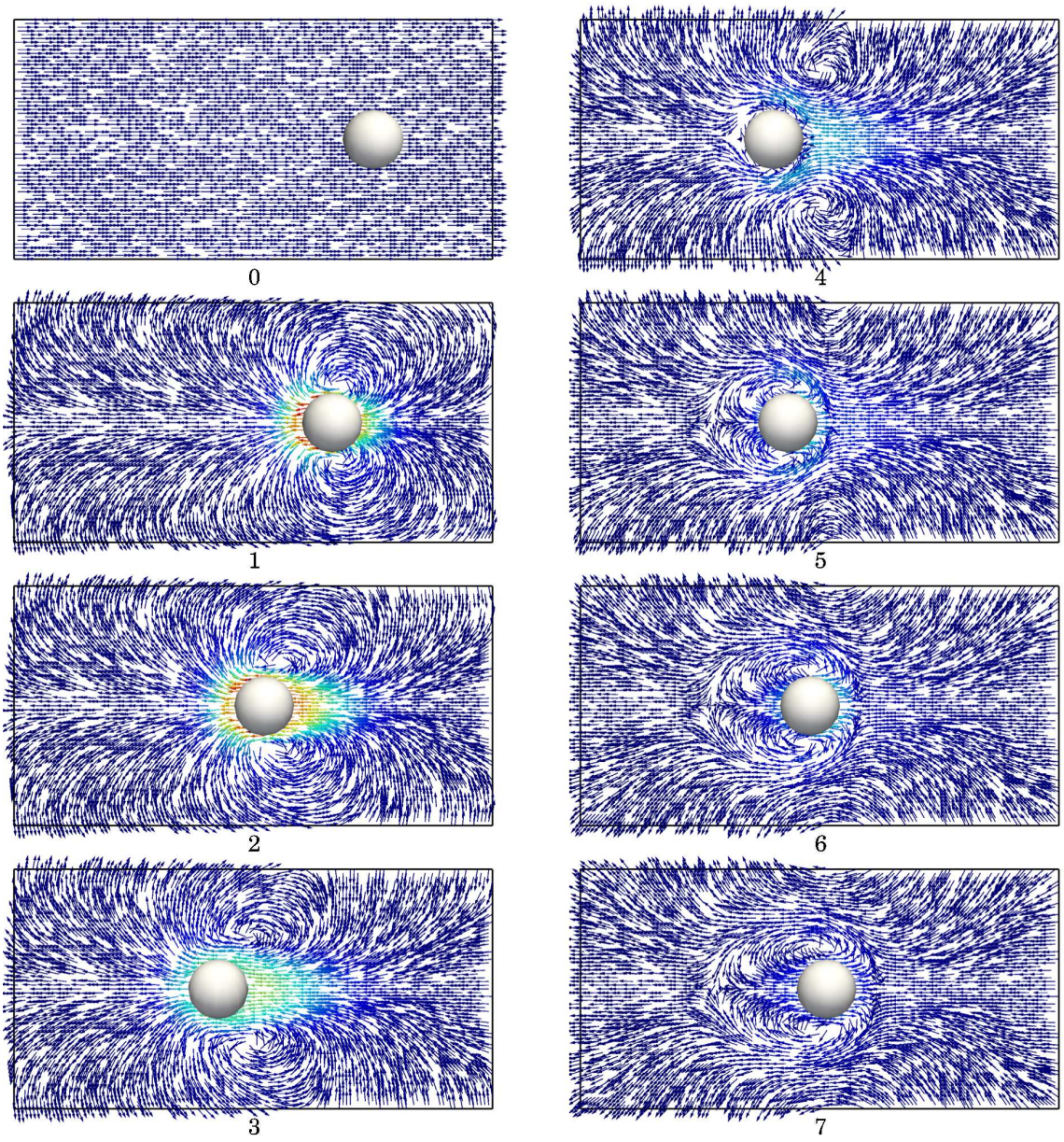


Figure 0.1: Oscillation of a sphere in a viscous fluid. The sphere is pulled towards the center with a linear spring. The arrows, colored by velocity magnitude, represent fluid velocity vectors. The sphere undergoes one oscillation in snapshots 0 through 7.

developing scientific applications for graphics processing units (GPUs) from a series of undergraduate internships with Dr. Matthew Lear and Mr. Michael Schwartz at the Johns Hopkins University Applied Physics Laboratory, I suggested that it could be the perfect time to develop a new code base designed specifically to leverage the devices that were becoming increasingly common in high performance computing systems. Professor Prosperetti agreed, marking the beginning of a journey more demanding—but also more rewarding—than either of us anticipated.

Eager to tidy up some of the more inelegant aspects of the previous implementation of the Physalis method, we began to develop new theory and numerical methods. We sought to improve upon crippling stability limitations while increasing the efficiency of the code. Each enhancement carried with it unique properties for us to discover, and the distinctive algorithmic rules required for effective GPU code deployment further increased the development workload. In this dissertation, we tell the story of this development process with attention to both the overarching concepts that hold the project together and the fine details that make it work.

To summarize the story, after four years of theoretical, numerical, algorithmic, and coding research and development, the Prosperetti research group operates a tool that runs up to 90 times faster than the code that I inherited. It now takes less than an hour to run precisely the same viscous harmonic oscillator simulation for which we once waited three weeks. We are capable of simulating thousands of particles dynamically interacting within an enveloping fluid in the same amount of time that it used to take

to simulate ten particles. This new tool has opened the door to the first-principles simulation of disperse particle flows at a scale and accuracy that will help improve the limited human understanding of the underlying physics that govern extremely important and complex phenomena from dust storms to air pollution control. While we recognize that single-GPU operation does not necessarily scale up to a many-GPU code, there are interesting phenomena that fit on a single GPU and for which our code is valuable. A case in point is the old viscous harmonic oscillator study that set this work in motion, which, despite being able to generate the entire data set in less than a day, we have yet to revisit.

We have released our tool, called Bluebottle, under an open source license, available at `http://PhysalisCFD.org`. It has been used by at least nine researchers at four different research institutions, has directly resulted in the funding of a National Science Foundation grant, and has generated, to date, two peer-reviewed journal articles with at least four more waiting in line. We hope to continue to support and expand the capabilities of Bluebottle and to share the results with any researcher interested in simulating disperse particle flows.

Acknowledgements

I would like to acknowledge first and foremost the tireless dedication, wisdom, excitement, and patience of my doctoral advisor Andrea Prosperetti, who took a big risk

in taking me on as a student. The National Science Foundation's Integrative Graduate Education and Research Traineeship (IGERT) Modeling Complex Systems fellowship program at the Johns Hopkins University (NSF grant DGE0801471), directed by Professor Lori Graham-Brady, made possible the vast majority of the development of Bluebottle and helped prepare me for multidisciplinary postdoctoral research. United States National Science Foundation grants CBET1258398 and CBET1335965 additionally supported this work. Many professors at the Johns Hopkins University have shaped me as a scholar, including Charles Meneveau, Rajat Mittal, Joseph Katz, Cila Herman, Gregory Eyink, Tamer Zaki, Dennice Gayme, Steven Marra, and countless others in the Whiting School of Engineering and the Krieger School of Arts & Sciences. I wish to particularly thank Professors Rajat Mittal and Tony Dalrymple for their time spent reviewing this dissertation. I appreciate the significant contributions of my coworkers in the Prosperetti research group (permanent and adopted) in improving Bluebottle: Laura Lukassen, Yayun Wang, Daniel Willen, Gedi Zhou, Yuhang Zhang, and Shigan Chu. Thank you to the administrative staff in the Mechanical Engineering Department for the work you do behind the scenes to facilitate the work of students like me: Mike Bernard, Marty Devaney, Kevin Adams, Deana Santoni, Barb Adamson, Cindy Larichiuta, and everyone else I have not had the opportunity to interact with directly. Thank you, Matt Lear, for convincing me that I would be a good fit in graduate school. Thank you Brett Newman and everyone else at Microway for helping me design the GPU development workstation and compute

cluster on which all of this work was completed.

I would not be the scientist, teacher, athlete, or person I am today without the endless love and support of my wife Claire VerHulst; thank you! Thank you to my parents, John and Terri, for putting me in a position to succeed in this endeavor, and to my siblings, Shane and Abbey, for helping me through the journey. Finally, thank you to my friends for supporting me, especially Paul Lee, John Nauman, and Adrien Thormann.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

We encounter disperse multiphase flows—where a continuous fluid phase (liquid or gas) carries a second phase (solid, liquid, or gas) dispersed within it—in countless applications in both nature and technology. These phenomena strongly affect human social and economic activities around the world. In nature, we readily find examples of multiphase flows, including rainfall (and formation), dust storms, avalanches, erosion, and even the lava flows and ash clouds of volcanic eruptions. Technologically, we find applications of multiphase flows in food and pharmaceutical production, the transport of materials in slurries, pollution filtering, and energy generation.

Considering only the tremendous complexity of the dynamic motion of uncountably many disperse entities strongly interacting with each other as well as the enveloping fluid, the collection of these various multiphase flows are extremely difficult to predict, forecast, control, or design. Additionally recognizing that these systems

often involve heat transfer, phase change, chemical reactions, electrostatics, and other phenomena, the richness of the physics governing multiphase flows becomes apparent (see, e.g., Crowe et al., 2012). In spite of the clear influence that disperse multiphase flows exert on our lives, we understand remarkably little about their behavior. To provide an idea of the complexity of even the most common phenomena—and not intending to devalue the contributions of the many talented scientists performing this research—consider that current theories are unable to correctly account for rain cloud formation (Grabowski and Wang, 2013) and that we are unable to model the wide variations encountered in erosion processes (Willgoose, 2005).

As a first step towards discussing disperse multiphase flows, we generally categorize them according to the phases involved, depending on the mechanisms accounting for interactions between the phases: bubbly flows (gas in liquid), flows with drops (liquid in gas/liquid), and particle flows (solid in gas/liquid). In this work, we focus specifically on disperse particle flows, where we assume the solid bodies to be essentially rigid so that they do not deform. This rigidity imparts the defining characteristics of interactions between each particle and the surrounding fluid, namely, the no-slip and no-penetration conditions at the surface of the particle. According to these conditions, the velocity of the air carrying a dust particle or the water transporting a sand grain must match that of the particle at all points on its surface. That is, should a heavy sand grain stand still at the bottom of a stream, the water flowing over it must slow to zero velocity along the entirety of the surface of the particle.

This idea is illustrated in Figure 1.1, in which the fluid velocity is seen to slow as it approaches the surface of a particle. Also note that the particles in Figure 1.1 can "feel" each other through the fluid, as a particle can "hide" from the full effect of the incoming flow if it sits in the wake of another particle. We will return to this

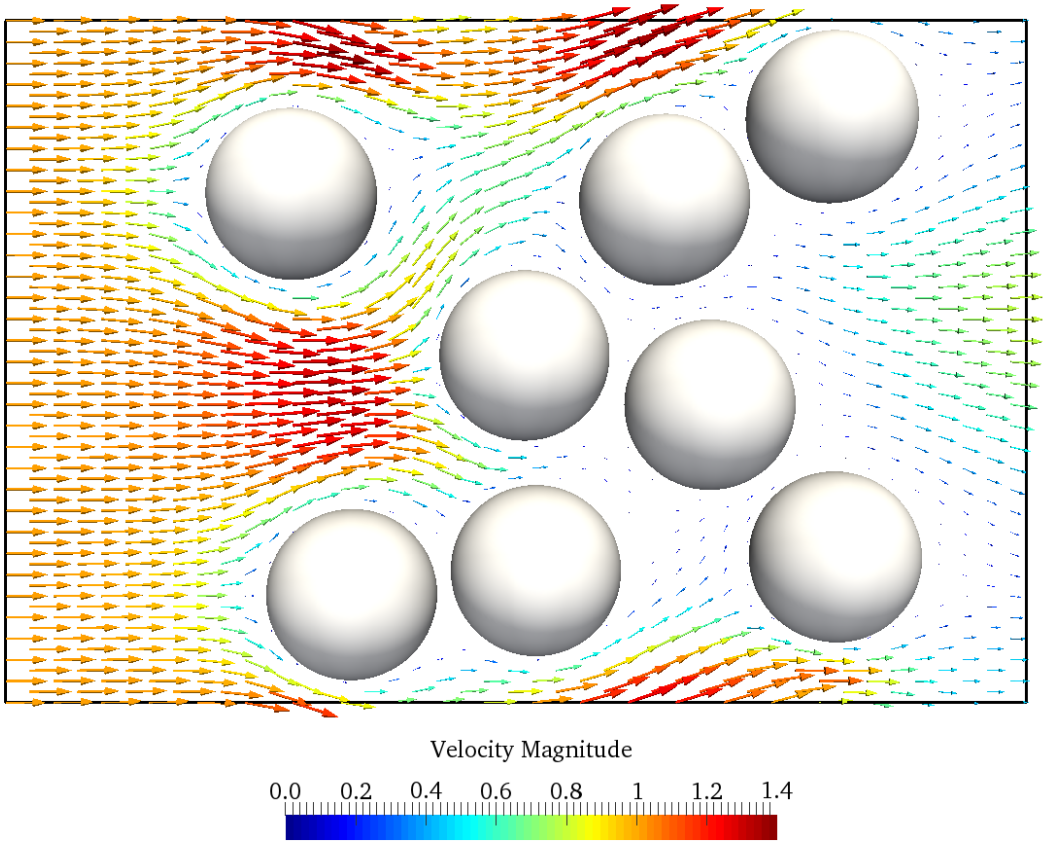


Figure 1.1: Flow through the midplane of an arrangement of immobile particles. The arrows represent velocity vectors normalized by the inflow velocity. Note that, in this three-dimensional simulation, fluid may flow freely out of the plane, so that that mass is not necessarily conserved in the plane (though it is conserved globally).

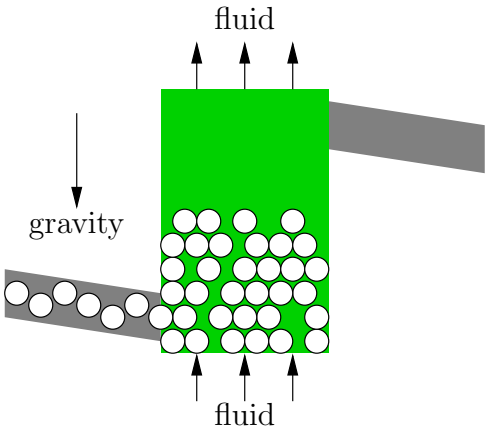
important effect shortly.

1.1 Disperse particle flows

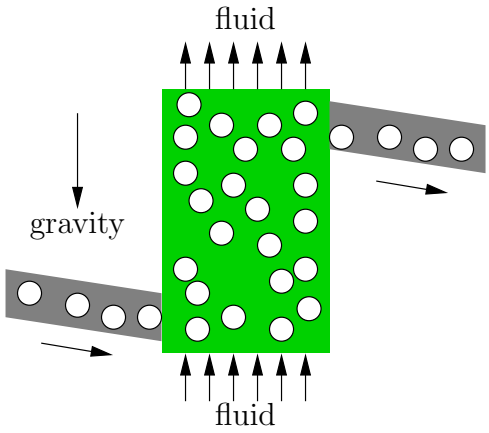
We have mentioned thus far a few examples of disperse particle flows, including dust storms, avalanches, erosion, transport slurries, and pollution filtering. Another important example of disperse particle flows, one worth discussing in more depth, finds numerous applications in technological processes: fluidized beds.

Consider a chemical reaction that occurs between a fluid (either liquid or gas) and a solid. Since the reaction takes place at the interface between the two materials, increasing the surface area of the solid immersed in the fluid will increase the overall reaction rate, suggesting that infiltrating the solid with the fluid could improve performance. Using a solid in particulate form is one convenient way of achieving this infiltration: by simply combining the two constituents, the fluid will naturally envelop the solid with extremely high surface area exposure. Assuming the solid has a higher density than the fluid (which is most often the case), the particles will settle to the bottom of a container as illustrated in Figure 1.2a, which is called a packed (or fixed) bed. As the chemical reaction proceeds, we can accelerate the reaction rate by supplying new unreacted fluid to the particle bed. A packed bed, by definition remaining stationary as the fluid streams through, will often experience extreme gradients in temperature and reacted constituent concentration as the random packing produces

uneven reaction rates throughout the bed. Further, as the catalytic particles become spent, the reaction rates decrease until the reaction slows to a point at which the particles must be replaced.



(a) A packed bed. When the particles have been spent, the reaction must be shut down in order to replace the particles.



(b) A fluidized bed. Spent particles may be replaced by new particles continuously without stopping the reaction.

Figure 1.2: By injecting fluid upward against gravity at different velocities, the drag force experienced by the particles determines the state of the particle bed.

Should we increase the velocity of the fluid stream, injected upward against gravity as depicted in Figure 1.2b, the drag experienced by the particles causes them to fluidize—i.e., to be advected about by the fluid instead of resting without motion—when the force exerted by the fluid stream exceeds the weight of the particles. This fluidized bed improves upon many of the limitations of the packed bed by significantly improving the mixing of the particles, which now move about the container. Improved

mixing tends to increase reaction rates by homogenizing reacted constituent concentration in the bed, and it also has the benefit of decreasing temperature gradients. Additionally, as illustrated in Figure 1.2b, the spent solid phase may be replaced without halting the reaction, further increasing yields of the chemical reaction over time (Saleh, 2002; Grace et al., 2006).

Because of their many benefits, fluidized beds find common application beyond chemical processes that benefit from fluidization, such as quick freezing of foodstuffs, bulk drying operations, and biomass or fossil fuel combustion. In all of these applications, the flow induced by the dynamic interaction of possibly trillions of particles in a single industrial-scale fluidized bed reactor is tremendously complex, comprised of flow structures and instabilities that vary widely depending on parameters such as the ratio of the densities of the particles and fluid, the size of the particle, and the fluid injection velocity (Sundaresan, 2003). The experimental work of van Buijtenen et al. (2011), summarized in Figure 1.3, illustrates a map of the many flow regimes of a pseudo-two-dimensional spout fluidized bed depending on the ratios of the velocities of the background flow u_{bg} and the spout u_{sp} with respect to the minimum fluidization velocity u_{mf}. Here, the bed of particles is fluidized either by the bulk flow or by the higher-velocity jets (or both), leading to highly complex macroscopic particle phase behavior.

The flow complexity inherent in fluidized beds, which occupy orders of magnitude less space than some naturally occurring disperse particle flows like dust storms or

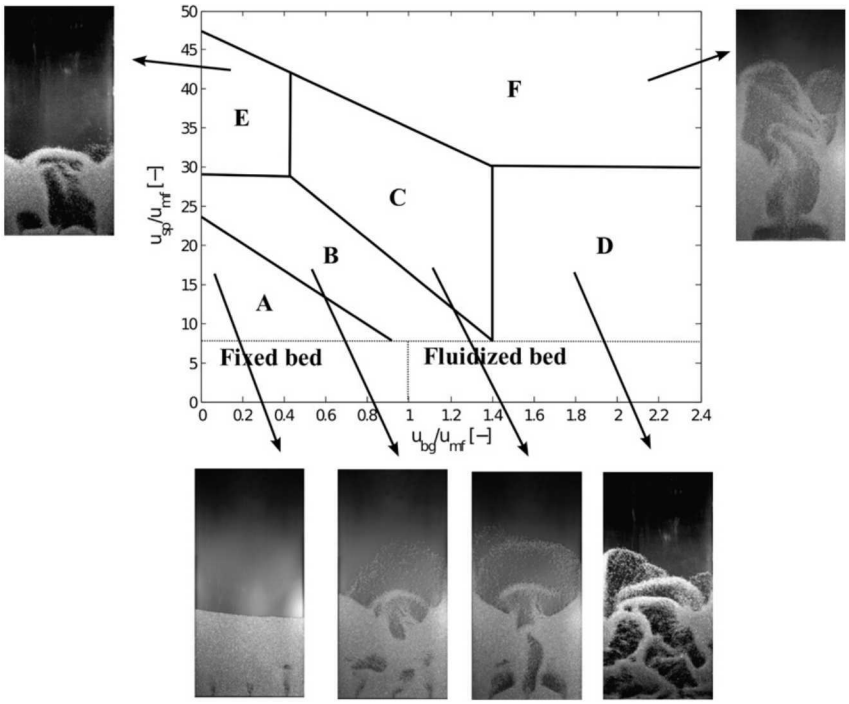


Figure 1.3: Fluidized bed regime map with example snapshots of a pseudo-two-dimensional spout fluidized bed, adapted from van Buijtenen et al. (2011). Here, u_{bg}, u_{sp}, and u_{mf} are the background, spout, and minimum fluidization velocities, respectively. The denoted flow regimes are: (A) multiple internal spouts; (B) multiple spouts; (C) multiple interacting spouts; (D) multiple jets in fluidized bed; (E) alternating two-spout contraction; and (F) contracted spouts with periodic channel blocking.

volcanic ash clouds, means that the majority of the design of industrial fluidized bed reactors is empirically based with limited first-principles knowledge of the physical processes governing their operation. Expensive and time consuming, the current

design process largely relies on prior experience and the construction and testing of large-scale model systems with some assistance from reduced order numerical models (Yang, 2003; Joshi and Nandakumar, 2015). The myriad industrial applications of fluidized beds—and, also, our ability to work with the many instances of disperse particle flows in general—would benefit significantly from an improved understanding of these physical processes.

The scientific investigation of disperse particle flows dates back to the work of Stokes (1851), and the literature to date is quite expansive (see, e.g., Leal, 1980; Stickel and Powell, 2005; Crowe et al., 2012), with much domain-specific knowledge among various applications, including earth sciences (see, e.g., Goudie, 2008), pollution control (see, e.g., Jorgensen and Johnsen, 1981), turbulent flows (see, e.g., Balachandar and Eaton, 2010), suspensions and porous media (see, e.g., Koch and Hill, 2001), transport and deposition (see, e.g., Guha, 2008), and many others. Significant advancements have been made by theoretically analyzing limit cases of the governing physics, for example in the low Reynolds number regime (Brady and Bossis, 1988) or the limit of infinitessimally small particles (Maxey, 1987), but away from those limit cases, we must resort to experimental and numerical techniques for probing the complex flow physics. The same complexity that limits purely theoretical progress also hinders the alternative methods of investigation, however.

As the primary topic of this dissertation, we will make abundantly clear the difficulties associated with numerical investigation techniques, but we should discuss

briefly the limitations of experimental observations. Consider the fluidized beds described above, for which it is quite reasonable to measure simple global parameters such as the pressure drop across the bed, the fraction of volume occupied by the particles, or the height of the bed when fluidized (Richardson and Zaki, 1954). However, recording more detailed microstructural data, such as particle kinematics, poses quite a challenge even in dilute particle concentrations (Katz and Sheng, 2010). Because fluidized beds generally experience rather large particle concentrations that block visual inspection, optical investigation techniques are typically limited to pseudo-two-dimensional beds like those considered in van Buijtenen et al. (2011). To obtain more realistic three-dimensional data in the bulk of a bed, researchers have resorted to exotic techniques like electrical capacitance tomography (Rautenbach et al., 2011) and X-ray tomography (Franka et al., 2007), but even these techniques record only the largest scales of particle motion. These strong experimental limitations have brought numerical techniques of modeling disperse particle flows to the forefront of current scientific exploration.

1.2 Modeling disperse particle flows

Generally speaking, the equations that govern disperse particle flows are well known. The incompressible Navier-Stokes equations,

$$\partial_t \mathbf{U} + (\mathbf{U} \cdot \nabla)\, \mathbf{U} = -\frac{1}{\rho_f} \nabla p + \nu \nabla^2 \mathbf{U} + \mathbf{g}, \tag{1.1a}$$

$$\nabla \cdot \mathbf{U} = 0, \tag{1.1b}$$

describe the momentum and mass conservation of the fluid phase at some time t with velocity $\mathbf{U}$, pressure p, density ρ_f, kinematic viscosity ν, and applied body force per unit mass $\mathbf{g}$. While tremendously rich in physics in their own right (see, e.g., Batchelor, 2000), the Navier-Stokes equations (1.1) tell only half of the story as they are subject to the dynamically evolving no-slip boundary conditions that must be applied at the surface of each of the uncountably many particles. As is well explained in Prosperetti (2015), the numerical modeling of disperse particle flows—and multiphase flows in general—faces the prospect of a certain "death by boundary conditions."

Under the influence of these boundary conditions, the linear momentum equation for a spherical particle α with radius a, mass ρ_p, volume $v = 4/3\pi a^3$, position $\mathbf{X}$, and velocity $\mathbf{w}$ is

$$\rho_p v \frac{d\mathbf{w}}{dt} = \oint_S dS\ \boldsymbol{\sigma} \cdot \hat{\mathbf{n}} + \sum_{\beta \neq \alpha} \mathbf{F}_{\alpha\beta} + \rho_p v \mathbf{g}, \tag{1.2}$$

The integral, accounting for the hydrodynamic force on the particle, is taken over the surface of the particle S with $\hat{\mathbf{n}}$ the outward normal with the incompressible viscous stress tensor

$$\boldsymbol{\sigma} = -p\mathbf{I} + \mu \left(\nabla \mathbf{U} + \nabla \mathbf{U}^T \right), \tag{1.3}$$

in which $\mathbf{I}$ is the identity two-tensor, $\mu = \rho_f \nu$ the dynamic viscosity, and the superscript T the transpose. The second term on the right-hand side of (1.2) represents

the force of particle β applied to particle α during direct contact and the third term incorporates an external force per unit mass $\mathbf{g}$. Likewise, the angular momentum equation for particle α with angular velocity $\mathbf{\Omega}$ is

$$I\frac{d\mathbf{\Omega}}{dt} = \oint_S dS\ \mathbf{r} \times \boldsymbol{\sigma} + \sum_{\beta \neq \alpha} \mathbf{L}_{\alpha\beta}, \tag{1.4}$$

where $I = 2\rho_p v a^2/5$ is the moment of inertia of the spherical particle, $\mathbf{r}$ is a radial location and $\mathbf{L}$ is a force due to direct contact. Unfortunately, knowing the governing equations, (1.1), (1.2), and (1.4), and knowing their solution in large, dynamic disperse particle systems are two very different things, and it is precisely the solution of this problem that we seek.

In the numerical modeling of the set of governing equations, there exists one ultimate limiting factor: the available computational capacity. The amount of available memory—both random access memory and hard disk storage space—limits the number of variables (or degrees of freedom) that may be considered together. The available processing capacity—both the speed and the number of processors available—limits the number of operations that may be performed in a specified amount of time. While regularly improving each year, these two limits place very real boundaries on the the size and scope of any numerical investigation that a scientist might hope to perform.

With trillions of particles with diameters on the order of millimeters or less, all strongly coupled to the surrounding fluid in an industrial fluidized bed reactor meters in height, even the top supercomputers in the world today are unable to store the entirety of a first-principles fluidized bed numerical simulation. Even if a computer

contained memory sufficient to store the required data, the requisite calculations would likely require many, many years to complete (Wilcox, 1998). Since computational expediency and physical approximation go hand in hand, significant effort has been put forth in order to simplify to various levels of accuracy the computational task by approximating (1.1) and the requisite boundary conditions (see, e.g., Prosperetti and Tryggvason, 2009; van der Hoef et al., 2008; Tenneti and Subramaniam, 2014; Fox, 2012).

Figure 1.4, adapted from Deen et al. (2007), compares two examples of reduced-order disperse particle flow simulation models to a similar experimental observation. The bottom and middle rows illustrate the results of so-called two-fluid and point-particle methods, respectively. Let us discuss the benefits and drawbacks of these methods in order to construct a hierarchy of modeling methods against which to compare the primary topic of this dissertation: resolved-particle direct numerical simulation.

1.2.1 Two-fluid models

The most highly simplified, the two-fluid models are the only methods capable of simulating an entire industrial-scale fluidized bed reactor (Joshi and Nandakumar, 2015). The name, two-fluid model, stems from the method's special treatment of the Navier-Stokes equations (1.1) and constituent particle evolution equations (1.2) and (1.4) that smooths the interacting disperse particles into a continuum. By avoiding

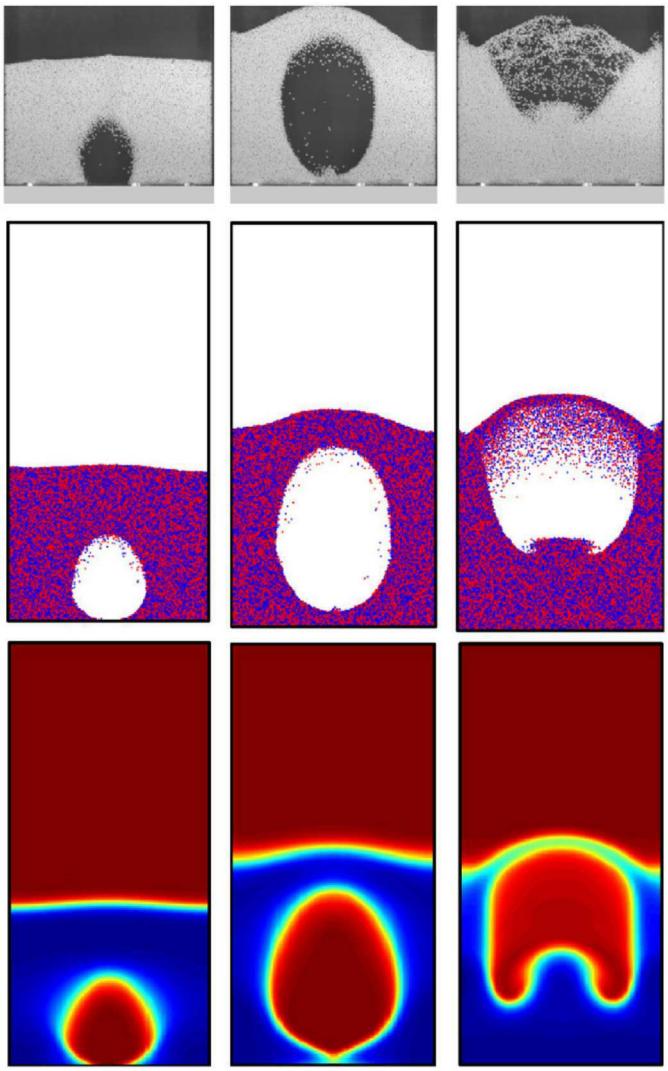


Figure 1.4: Comparison of three snapshots in time of an experiment (top) with point-particle (middle) and two-fluid (bottom) simulations, adapted from Deen et al. (2007).

the direct inclusion of the particle boundary conditions, these methods forgo tracking the motion of any particles at all, instead relying on the interaction of two continuous phases—the real fluid phase and the pseudo-fluid particle phase—to model the system.

There are many ways to average the governing equations to simplify the solution process, including time averaging and volume averaging (Crowe et al., 2012). Focusing on models capable of evolving in time, we summarize here a most basic volume averaging two-fluid model, adapted from Jackson (2000).

Define the local space average in terms of a monotonically decreasing weighting function $g(r)$, where r is the radial distance from the point in question, normalized so that

$$4\pi \int_0^\infty dr\ g(r)\, r^2 = 1. \tag{1.5}$$

We choose a g that satisfies the "soft" average constraint assuming g is differentiable as many times as needed and defining an averaging radius l such that

$$\int_0^l dr\ g(r)\, r^2 = \int_l^\infty dr\ g(r)\, r^2. \tag{1.6}$$

The average value of any point function f at position $\mathbf{x}$ and time t is therefore defined as

$$\langle f \rangle (\mathbf{x}, t) = \int_V dV_y\ f(\mathbf{y}, t)\, g(|\mathbf{x} - \mathbf{y}|) \tag{1.7}$$

where V is the volume of the entire system. If this volume containing particles of size a may be characterized by a length L, the amount of error introduced by the averaging process depends on the magnitude of the separation of these scales; the more strongly $L \gg l \gg a$, the less sensitive the local averaging becomes with respect to the particular choice of g. This separation of scales may be difficult to justify in some disperse particle flows, as Laín et al. (2002) indicated that the microscale particle

interactions can strongly influence the overall system behavior even for dilute particle concentrations.

We may take averages of the fluid phase such as the void fraction,

$$\varepsilon(\mathbf{x}) = \int_{V_f} dV_y \; g(|\mathbf{x}-\mathbf{y}|), \tag{1.8}$$

and the fluid phase average

$$\langle f \rangle^f(\mathbf{x}) = \frac{1}{\varepsilon(\mathbf{x})} \int_{V_f} dV_y \; f(\mathbf{y})\, g(|\mathbf{x}-\mathbf{y}|). \tag{1.9}$$

In a similar way, henceforth assuming identical spherical particles of radius a and mass ρ_p, we integrate over the disjoint region of particle interiors to define the particle number density

$$n(\mathbf{x}) = \sum_{\alpha} g(|\mathbf{x}-\mathbf{X}_\alpha|), \tag{1.10}$$

the particle volume fraction

$$\phi(\mathbf{x}) = \frac{4}{3}\pi a^3\, n(\mathbf{x}), \tag{1.11}$$

and the particle phase average

$$\langle f \rangle^p(\mathbf{x}) = \frac{1}{n(\mathbf{x})} \sum_{\alpha} f^\alpha\, g(|\mathbf{x}-\mathbf{X}_\alpha|) \tag{1.12}$$

where $\mathbf{X}_\alpha$ is the location of the α^{th} particle center. With these definitions, we can see that the overall average (1.7) is related to the phase averages by

$$\langle f \rangle = \varepsilon \langle f \rangle^f + \phi \langle f \rangle^p, \tag{1.13}$$

and we can additionally define the mass average $\langle f \rangle^m$ by

$$\bar{\rho}\langle f \rangle^m = \rho_f \varepsilon \langle f \rangle^f + \rho_p \phi \langle f \rangle^p, \tag{1.14}$$

where $\bar{\rho} = \rho_f \varepsilon + \rho_p \phi$.

By applying these definitions to the equation of mass conservation (1.1b), we find that

$$\frac{\partial \varepsilon}{\partial t} + \frac{\partial}{\partial x_k}\left(\varepsilon \langle U_k \rangle^f\right) = 0, \tag{1.15}$$

where we temporarily adopt for the remainder of this section the Einstein summation convention using the subscript $k = \{1, 2, 3\}$ to indicate the vector component. Additionally, we may form the continuity equation for the particle phase

$$\frac{\partial n}{\partial t} + \frac{\partial}{\partial x_k}\left(n \langle U_k \rangle^p\right) = 0. \tag{1.16}$$

Working now with momentum equation (1.1a), we find the fluid phase momentum conservation equation (see, e.g., Jackson, 2000)

$$\begin{aligned} &\rho_f \left[\frac{\partial}{\partial t}\left(\varepsilon \langle U_i \rangle^f\right) + \frac{\partial}{\partial x_k}\left(\varepsilon \langle U_i U_k \rangle^f\right)\right] \\ &\quad = \frac{\partial}{\partial x_k}\left(\varepsilon \langle \sigma_{ik} \rangle^f\right) - \sum_\alpha \oint_S dS_y \; \sigma_{ik}\left(\mathbf{y}\right) n_k\left(\mathbf{y}\right) g\left(|\mathbf{x} - \mathbf{y}|\right) + \rho_f \varepsilon g_i. \end{aligned} \tag{1.17}$$

In a similar way, the particle linear momentum equation (1.2) yields the averaged particle phase linear momentum equation,

$$\begin{aligned} &\rho_p v \left[\frac{\partial}{\partial t}\left(n \langle U_i \rangle^p\right) + \frac{\partial}{\partial x_k}\left(n \langle U_i U_k \rangle^p\right)\right] \\ &\quad = \sum_\alpha g\left(|\mathbf{x} - \mathbf{X}_\alpha|\right) \left[\oint_S dS_y \; \sigma_{ik}\left(\mathbf{y}\right) n_k\left(\mathbf{y}\right) + \sum_{\beta \neq \alpha} F_i^{\alpha\beta}\right] + \rho_p v n g_i, \end{aligned} \tag{1.18}$$

and the particle angular momentum equation (1.4) yields the averaged particle phase angular momentum equation,

$$I\left[\frac{\partial}{\partial t}\left(n\langle\Omega_i\rangle^p\right)+\frac{\partial}{\partial x_k}\left(n\langle\Omega_i U_k\rangle^p\right)\right] = a\,\varepsilon_{ilm}\sum_{\alpha} g\left(|\mathbf{x}-\mathbf{X}_\alpha|\right)\left[\oint_S dS_y\;\sigma_{mk}\left(\mathbf{y}\right)n_l\left(\mathbf{y}\right)n_k\left(\mathbf{y}\right)+\sum_{\beta\neq\alpha}L_m^{\alpha\beta}\right], \tag{1.19}$$

where ε_{ilm} is the Levi-Civita symbol.

The set of equations (1.15) through (1.19) may be used together to evolve the system through time. Note carefully, however, the terms in (1.15) through (1.19) containing $\langle U_i U_k\rangle^f$, $\langle\sigma_{ik}\rangle^f$, $\langle U_i U_k\rangle^p$, $\langle\Omega_i U_k\rangle^p$, and all of the surface integrals, none of which are we able to close analytically. These unclosed terms account for the effect of the fluid-fluid, fluid-particle, and particle-particle interactions that we have averaged away. While much literature exists on the topic of the development of closure models (see, e.g., Jackson, 2000)—such as those that have been posed for large and small Stokes number, $\mathrm{St} = (1/9)\,(2aw_r/\nu)\,(\rho_p/\rho_f)$, based on the relative particle approach velocity w_r, as well as some other limiting cases—we can draw the following conclusions about their effectiveness: Some closures do some things well; most exhibit anomalous behavior within their ranges of validity; and all fail when exceeding their ranges of validity (Jackson, 2000). The true limitation of these two-fluid models, therefore, is our limited ability to accurately model the unclosed terms that appear as a result of the averaging process.

1.2.2 Point-particle models

To decrease the amount of approximation and thereby increase model accuracy, we consider the so-called point-particle models. Unlike two-fluid models, point-particle models track each individual particle in the simulation, but impart to them infinitesimal extent. In this way, point-particle models achieve increased fidelity by resolving more accurately the effects of individual particles on the fluid. The method pays for the increased fidelity with a decreased physical domain size, which is limited to a moderate fraction of an industrial-scale fluidized bed reactor (Crowe et al., 2012). As the literature is quite extensive, we briefly summarize here the basic theory of point-particle methods, adapted from Prosperetti and Tryggvason (2009).

Point-particle models solve the Navier-Stokes equations as written in (1.1) where $\mathbf{g}$ takes the form

$$\mathbf{g} = \mathbf{g}' - \frac{\mathbf{F}}{\rho_f}, \tag{1.20}$$

where $\mathbf{g}'$ is an applied body force per unit mass and $\mathbf{F}$ is a force per unit volume exerted by the particles on the fluid, approximated by a superposition of Dirac delta functions

$$\mathbf{F} = \sum_{\alpha} \mathbf{f}_\alpha \delta \left(\mathbf{x} - \mathbf{X}_\alpha \right). \tag{1.21}$$

Here, the form of the force $\mathbf{f}_\alpha$ of particle α on the fluid is unknown and must be modeled.

Most often, $\mathbf{f}_\alpha$ is imparted the form

$$\mathbf{f}_\alpha = \rho_f v \left(\frac{D\mathbf{U}_\infty}{Dt} - \mathbf{g}' \right) + \mathbf{f}_d + \mathbf{f}_a + \mathbf{f}_h + \mathbf{f}_{\text{additional}}, \tag{1.22}$$

using characteristic far-field velocity $\mathbf{U}_\infty$ where

$$\frac{D\mathbf{U}_\infty}{Dt} = \frac{\partial \mathbf{U}_\infty}{\partial t} + (\mathbf{U} \cdot \nabla)\, \mathbf{U}_\infty \tag{1.23}$$

and the various $\mathbf{f}_*$ are modeled effects that combine analytical results for limiting cases with additional empirical considerations. The drag force $\mathbf{f}_d$, generally the leading-order effect, often takes the form

$$\mathbf{f}_d = \frac{1}{2} \rho A C_d \left| \mathbf{U}_\infty - \mathbf{w} \right| (\mathbf{U}_\infty - \mathbf{w}), \tag{1.24}$$

where $A = \pi a^2$ is the frontal area of a sphere and the drag coefficient on a sphere up to a particle Reynolds number $\mathrm{Re}_p = 2a \left| \mathbf{U}_\infty - \mathbf{w} \right| / \nu$ up to 1000 is well estimated by

$$C_d = \frac{24}{\mathrm{Re}_p} \left(1 + 0.15 \mathrm{Re}_p^{0.687} \right). \tag{1.25}$$

The relation (1.24) performs quite well in situations similar to that for which it was derived—namely, the uniform flow of fluid with velocity $\mathbf{U}_\infty$ far away and no other particles nearby—and it is clear that the accuracy of (1.24) degrades rapidly as local particle concentrations increase beyond the dilute limit. Revisiting Figure 1.1 on page 3 as an example, we can expect that (1.24) is likely to perform moderately well for the particle nearest the inflow on the left side of the domain, but that the accuracy of (1.24) for the other particles, strongly influenced by the presence of the

neighbors nearby, is likely to suffer severely. This consideration opens a number of questions, such as: How close is too close for two particles? and What velocity should we choose for $\mathbf{U}_\infty$? All of these effects limit the accuracy of point-particle methods. The other forces collected in (1.22) include the added mass force

$$\mathbf{f}_a = \frac{1}{2}\rho_f v \left(\frac{D\mathbf{U}_\infty}{Dt} - \frac{d\mathbf{w}}{dt} \right), \tag{1.26}$$

the history force

$$\mathbf{f}_h = 6a^2\rho_f\sqrt{\pi\nu}\int_{t_0}^{t} \frac{dt'}{(t-t')^{1/2}} \left(\frac{D\mathbf{U}_\infty}{Dt'} - \frac{d\mathbf{w}}{dt'} \right), \tag{1.27}$$

and any additional forces like collisions, electrostatic interactions, or lift due to shearing flows.

So far, we have applied the force of the particle on the fluid. To apply the influence of the fluid on the particle, simply use the equation of motion (1.2) and approximate the integral term with the modeled force, namely, $\mathbf{f}_\alpha$ (1.22). Point-particle methods, with various assumptions made for the form of the particle forces, have been applied extensively in the literature to gain much insight into the evolution of some types of disperse particle flows (see, e.g., Maxey, 1987; Elghobashi and Truesdell, 1992; Sundaram and Collins, 1996; Prosperetti and Tryggvason, 2009; Crowe et al., 2012). Limitations to their physical accuracy, however, clearly reside in the approximations made in modeling $\mathbf{f}_\alpha$ (1.22).

1.2.3 Resolved-particle simulation

To further improve accuracy, we consider the primary topic of this dissertation: resolved-particle simulation. We present here a brief review of the resolved-particle literature and develop one such method, the Physalis method, in Chapter 2.

As illustrated in Figure 1.1, resolved-particle simulations are characterized by the numerical resolution of the no-slip boundary condition at the surface of the particle. In this way, the fluid and particle equations (1.1), (1.2), and (1.4) are coupled at a first-principles level without the modeling required in both the two-fluid and the point-particle methods. The price to pay for this accuracy, however, is the significant increase in numerical complexity that limits the scope of the simulation to a small fraction of an industrial-scale fluidized bed reactor.

A landmark contribution to particle-resolved simulations was made by Uhlmann (2008), who was able to simulate channel flow of a fluid with 4096 suspended particles by means of an immersed boundary method that he had developed earlier (Uhlmann, 2005). The immersed boundary method, first posed in Peskin (1977), applies the influence of an immersed body to the fluids equations. In general, there are two methods for achieving this:

1. Discretize the fluid phase in a way that conforms to the shape of the immersed body;

2. Discretize the fluid phase with a simple grid and account for the geometrical

mismatch between the grid and the shape of the immersed body.

When simulating an immersed body that may be transformed into a reference frame in which it is stationary, method 1 is preferred because it provides the most direct specification of the fluid/body interface and also allows for the space to be discretized with higher accuracy in regions of particular interest. When simulating an immersed body that may not be transformed in such a way—when two bodies are moving in opposite directions, e.g.—method 1 becomes less attractive because of the computational effort required to regenerate the discretization mesh every time a body moves (Tezduyar, 2001). To combat this limitation, the immersed boundary method follows method 2, solving the fluid equations (1.1) on a fixed grid as the body immersed in the fluid moves about the domain. In order to account for the geometrical mismatch between the grid and the body, the immersed boundary method essentially determines the force required for the fluid to accommodate the body, which is included in the force $\mathbf{g}$ in (1.1). Many ways of implementing this general idea have been developed, and Mittal and Iaccarino (2005) provides a more probing introduction to the method.

One particular limitation of the method posed by Uhlmann (2005) is the requirement to track the surface of the particle, which typically demands the evolution of about 1,000 Lagrangian tracking points for each particle Kempe and Fröhlich (2012a). We will see in Chapter 2 that there exist ways of overcoming this requirement using only 25 pieces of data for each particle.

In the years following Uhlmann (2008), the same method or some variation thereof

was used by several researchers. Lucci et al. (2010) presented results of the simulation of decaying homogeneous isotropic turbulence with up to 6400 particles. Kempe and Fröhlich (2012a) and Kempe and Fröhlich (2012b) improved the satisfaction of the no-slip condition and the collision algorithm of Uhlmann (2005). Picano et al. (2015) used a similar method to investigate 10,000 neutrally buoyant spheres in a channel flow. In the most recent simulations, Kidanemariam and Uhlmann (2014) have been able to simulate more than 260,000 particles.

Somewhat different versions of the immersed boundary method have been pursued by other researchers including Mehrabadi et al. (2015), Yang and Stern (2015), and Ji et al. (2014). Entirely different approaches have also been developed, such as the lattice Boltzmann method (see, e.g. Wylie et al., 2003; Sarkar et al., 2009), the fictitious domain method (see, e.g. Apte et al., 2009; Doostmohammadi and Ardekani, 2015) and its variants, the spectral element method (see, e.g. Zeng et al., 2010), and the pressure boundary integral method of Simeonov and Calantoni (2012).

1.3 Current contribution

It is clear that resolved-particle simulation tools may be used to improve the accuracy of the closure models used in the two-fluid and point-particle reduced order models discussed above (van der Hoef et al., 2008), though the precise manner in which to achieve this goal is far from clear. In this dissertation, we discuss for

the purpose of improving upon current reduced-order closure models a new resolved-particle simulation tool based on the Physalis method (Zhang and Prosperetti, 2005; Gudmundsson and Prosperetti, 2013) for the numerical solution of the disperse particle flow evolution equations (1.1), (1.2), and (1.4) for thousands of particles. This method introduces the influence of spherical particles to a fixed-grid incompressible Navier-Stokes flow solver using a local analytic solution to the flow equations. We discuss in Chapter 2 numerous enhancements to the efficiency and stability of the method.

Our first-principles investigation demands the modeling of unresolved length and time scales associated with particle collisions, which we consider in Chapter 3. We introduce a collision model alongside Physalis, incorporating lubrication effects and proposing a new nonlinearly damped Hertzian contact model.

In Chapter 4, we discuss the implementation of Physalis for massively parallel computation using a graphics processing unit (GPU). We combine Eulerian grid-based algorithms with Lagrangian particle-based algorithms to achieve computational throughput up to 90 times faster than the legacy implementation of Physalis for a single central processing unit (CPU). By avoiding all data communication between the GPU and the host system during the simulation, we utilize with great efficacy the GPU hardware with which many high performance computing systems are currently equipped.

In Chapter 5, by performing self-consistency simulations and reproducing experi-

mental studies from the literature, we document extensive validation of the methods. Chapter 6 briefly shows some potential capabilities of the work in various applications. Finally, we conclude in Chapter 7 by looking forward to the future of Physalis with multi-GPU parallelization in order to perform resolved disperse flow simulations of more than 100,000 particles and further advance the development of reduced-order closure models.

The majority of the discussion presented below has been adapted from work published in (or destined for) peer-reviewed scientific journals. The theoretical enhancements of Chapter 2 were published in Sierakowski and Prosperetti (2016). The new contact model of Chapter 3 is the subject of a forthcoming paper (Sierakowski, 2016b). Chapter 4 largely contains the work of Sierakowski (2016a), currently in press. The validation and applications discussion is formed from a combination of results from all three of the aforementioned papers as well as contributions from coworkers in the Prosperetti research group. All of the work presented here was funded by the United States National Science Foundation under grants DGE0801471, CBET1258398, and CBET1335965.

Chapter 2

The Physalis method

In brief, the Physalis method is characterized by the procedure used to transfer the no-slip boundary conditions satisfied at the surface of a spherical particle to an underlying fixed grid. It is, in that way, similar to the immersed boundary method discussed in Section 1.2.3. The primary difference lies in the manner by which Physalis communicates the effect of a particle to the fluid: Instead of applying forces in 1.1a, Physalis applies velocity and pressure boundary conditions. The procedure has many benefits:

1. The geometry of each particle is not approximated: Physalis maintains a sharp interface and satisfies the no-slip boundary condition to analytical accuracy;

2. The particle surface is defined implicitly while tracking only its position with no Lagrangian surface tracking required;

3. The numerical error decreases exponentially with the number of degrees of freedom representing each particle (i.e., Lamb's coefficients; see Section 2.4), which permits the use of relatively coarse grids without compromising numerical accuracy;

4. The hydrodynamic force, couple, and higher-order multipoles are generated directly in the course of the solution procedure with no need for additional calculations.

Its primary drawback, however, is that it may only be applied to particles for which we have analytic local flow solutions, namely spheres (and, perhaps, ellipsoids, though it has yet to be attempted).

In this chapter, we thoroughly develop the underlying theory of the Physalis method, which was first put forth in Prosperetti and Oğuz (2001) for cylinders in a two-dimensional potential flow. Takagi et al. (2003) extended the method to solve the incompressible Navier-Stokes equations in two dimensions. Physialis matured into a modern research tool in Zhang and Prosperetti (2005), which was the first version capable of simulating spherical particles in an incompressible viscous fluid. Since then, it has seen application in a handful of physical studies in Zhang and Prosperetti (2009), Liu and Prosperetti (2010), Naso and Prosperetti (2010), Liu and Prosperetti (2011), and Botto and Prosperetti (2012). Physalis was always intended to accurately and efficiently simulate large numbers of particles, but only Zhang et al. (2006) exhibited this potential. Unfortunately the same qualities that afford Physalis outstanding

fluid-particle boundary condition coupling demand a computational load limiting the simulation of many particles and/or a significant number of ensemble realizations. A marginal improvement to this efficiency limitation was put forth in Gudmundsson and Prosperetti (2013), but even then it was recognized that the method lacked the numerical efficiency required for truly large-scale particle simulations. This dissertation gathers together a number of enhancements to the Physalis method, the component flow solver, and their respective implementations that have significantly improved the capabilities of the tool.

As the method is rather intricate, we begin with a quick overview by describing the flow solution procedure at each point in time. Given the solution from the previous time step computed using a fairly standard second-order finite difference projection method on a fixed, staggered Cartesian grid (detailed in Section 2.1), we construct a particle *cage*—the subset of discrete flow solver cells at which a particle's influence is communicated—based on each particle's current position (see Section 2.2). Each particle's influence is communicated by analytically posing velocity and pressure boundary conditions—$\tilde{\mathbf{u}}$ and $\tilde{p}$, respectively, transformed to the particle rest frame from $\mathbf{U}$ and p (see Section 2.3)—on the cage using *Lamb's solution* to Stokes flow about a sphere (see Section 2.4). Due to the no-slip boundary condition at the surface of each particle, Lamb's solution increases in accuracy as the distance from the surface decreases. We communicate flow conditions from the global Navier-Stokes solution to the local Lamb's solution through *Lamb's coefficients* (Section 2.5). We

then solve the Navier-Stokes equations (1.1) according to the boundary conditions specified by Lamb's solution (see Section 2.6) and check that the two solutions match at their interface. If the flow conditions are changing rapidly between subsequent time steps, the local Lamb's solution is unlikely to properly match the global Navier-Stokes solution on the first attempt, and the process must be iterated until the two solutions converge. The iterative nature of the Physalis method requires multiple solutions of the Navier-Stokes equations (1.1) for every time step (typically one to ten, depending on flow conditions), which is the primary source of its heavy computational load. Upon convergence, the global flow field matches with each local particle flow field and we obtain the hydrodynamic forces directly from Lamb's coefficients using only a simple algebraic relation (see Section 2.7). We then integrate particle motion forward in time using the hydrodynamic, interaction, and any other forces present. Finally, we proceed to the next time step and repeat the process. The rationale, implementation, and consequences of these procedures are discussed in detail Section 2.8.

Now we present the details of the method as applied in this work, which we frequently contrast to the methods of Zhang and Prosperetti (2005) and Gudmundsson and Prosperetti (2013)—henceforth referred to as Z&P and G&P, respectively. For a clear delineation of the differences between the present and former methods, refer to Sierakowski and Prosperetti (2016), from which the present discussion was adapted.

2.1 The flow solver

Although Physalis may be attached to any generic flow solver, the interaction between the two is rather complex and we can most clearly explain the details of the Physalis method itself if we first set the stage by discussing the flow solver. The flow solver adopted in this work uses a fairly standard second-order finite difference projection method in a staggered arrangement on a uniform Cartesian grid for solving the incompressible Navier-Stokes equations (1.1) (see Kim and Moin, 1985; Ferziger and Perić, 2002, for more discussion). While the work presented in Z&P claimed to use fully second-order numerical schemes, we have recognized that the pressure-free formulation used to solve the Navier-Stokes equations achieves only first-order accuracy for pressure (Brown et al., 2001) and have improved the scheme to realize second-order accuracy for both pressure and velocity in the current work.

To proceed from time t^n to time $t^{n+1} = t^n + \Delta t$, the finite-difference discretized Navier-Stokes equations take the form

$$\frac{\mathbf{U}^* - \mathbf{U}^n}{\Delta t} = -\left[\left(\mathbf{U}\cdot\nabla_h\right)\mathbf{U}\right]^{n+1/2} - \frac{1}{\rho_f}\nabla_h p^{n-1/2} + \nu\nabla_h^2\mathbf{U}^{n+1/2} \tag{2.1}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^* - \frac{\Delta t}{\rho_f}\nabla_h\phi^{n+1} \tag{2.2}$$

where the subscript h indicates second-order central finite-difference derivatives and the star superscript indicates an operator-split intermediate velocity. The boundary conditions on $\mathbf{U}^*$ are the same as those on $\mathbf{U}^n$. An untraditional choice motivated in

Section 2.1.1, we treat both the convective and diffusive terms in (2.1) in an explicit manner using the variable-time second-order formulation of the Adams-Bashforth method: Taking $\mathbf{D}^n$ to represent either the convective or diffusive derivative at time t^n, we advance the derivative terms using the formula

$$\mathbf{D}^{n+1/2} = \left(1 + \frac{1}{2}\frac{\Delta t}{\Delta t_{-1}}\right)\mathbf{D}^n - \frac{1}{2}\frac{\Delta t}{\Delta t_{-1}}\mathbf{D}^{n-1} \tag{2.3}$$

where $\Delta t_{-1} = t^n - t^{n-1}$ and $\Delta t = t^{n+1} - t^n$ are previous and current time step sizes, respectively. We present our method for adaptively determining the size of the next time step in Section 2.1.2.

To enforce the continuity equation (1.1b) at $\mathbf{U}^{n+1}$, taking the divergence of (2.2) provides the following differential Poisson equation

$$\nabla_h^2 \phi^{n+1} = \frac{\rho_f}{\Delta t}\nabla_h \cdot \mathbf{U}^*, \tag{2.4}$$

with external boundary conditions $\mathbf{n} \cdot \nabla_h \phi^{n+1} = 0$. Previously, Z&P used a fast Fourier solver to find ϕ, but a fast solver cannot recognize the particle boundary conditions posed by Physalis. To overcome this difficulty, Z&P used an iterative deferred correction scheme for every sub-time-step (Lamb's coefficient) iteration, which failed to completely decouple the flow solutions across the cage and also proved to be rather inefficient. To achieve a complete decoupling between the interior and exterior of the particles—using a technique expounded upon in Section 4.3.4—we have adopted a Jacobi-preconditioned conjugate gradient solver in place of the fast Fourier solver. As a consequence we also ensure that our Physalis boundary conditions, described in

Section 2.6 are more accurately satisfied.

After solving for ϕ^{n+1} in (2.4), we complete the time step by projecting $\mathbf{U}^*$ from (2.1) onto a divergence-free space using (2.2) to satisfy (1.1b). The pressure is then updated using the solution for ϕ^{n+1} via

$$p^{n+1/2} = p^{n-1/2} + \phi^{n+1}. \tag{2.5}$$

2.1.1 Explicit numerics

Treating the diffusive term implicitly (e.g., using a Crank-Nicolson method) removes the diffusive stability requirement so that only the convective stability requirement (CFL condition) remains, resulting in an upper bound for the time step given by

$$\Delta t_I < \frac{\Delta x}{\max|U|},$$

written here in one dimension for simplicity; Δx is the discretization length. According to Ferziger and Perić (2002, p. 146), the stability limit on an explicit method is instead

$$\Delta t_E < \frac{1}{\dfrac{\max|U|}{\Delta x} + \dfrac{2\nu}{(\Delta x)^2}} = \frac{\Delta t_I}{1 + 2\mathrm{Re}_c^{-1}}, \tag{2.6}$$

where

$$\mathrm{Re}_c = \frac{\max|U|\,\Delta x}{\nu} = \frac{\Delta x}{2a}\mathrm{Re}_p, \tag{2.7}$$

is the cell Reynolds number and $\mathrm{Re}_p = 2a\max|U|/\nu$ is the particle Reynolds number. We note that (2.6) ensures the simultaneous satisfaction of both the convective

stability limit $\Delta t < \Delta x / \max|U|$ and the diffusive stability limit $\Delta t < (\Delta x)^2 / (2\nu)$. If Re_c is of order one or greater, the difference between implicit and explicit bounds is small, which implies that the explicit method becomes more efficient because it avoids the solution of a Helmholtz equation. In our simulation, typically $a/\Delta x \approx 8$ so that we find $\mathrm{Re}_c \approx 1$ when $\mathrm{Re}_p \approx 20$. We have verified that the expectation suggested by this argument is supported by numerical evidence for Re_p greater than this value.

Strictly in terms of time step size, the explicit method becomes less competitive for smaller Re_p. However, this disadvantage is at least partially compensated by the increased rate of convergence of Lamb's coefficient iterations as the time step is reduced. Further, the explicit method avoids the splitting error of order $O\left[(\Delta t)^2/\mathrm{Re}\right]$ for velocity associated with the implicit fractional step method, ensuring accuracy in unsteady simulations at all nonvanishing Reynolds numbers. Other methods, designed for solving the Navier-Stokes equations at vanishing Re_p, may prove more efficient, but the explicit method adopted here guarantees both stable and accurate time advancement for any nonvanishing Reynolds number, limited only by particle resolution.

2.1.2 Adaptive time advancement

By adapting the time step size to the time-dependent flow conditions, we proceed through the simulation significantly more efficiently than if we were restricted to choosing the largest stable time step size for the entire duration of the simulation. At

the end of every time step, we search the velocity field for the largest value in each of the three velocity components $\max|U_i|$, where the subscript i indicates $\{x, y, z\}$. We then select the next time step abiding by the convective (CFL) and diffusive stability criterion according to Ferziger and Perić (2002):

$$\Delta t = \frac{C}{\sum_i \left[\frac{\max|u_i|}{\Delta x_i} + \frac{2\nu}{(\Delta x_i)^2} \right]}, \tag{2.8}$$

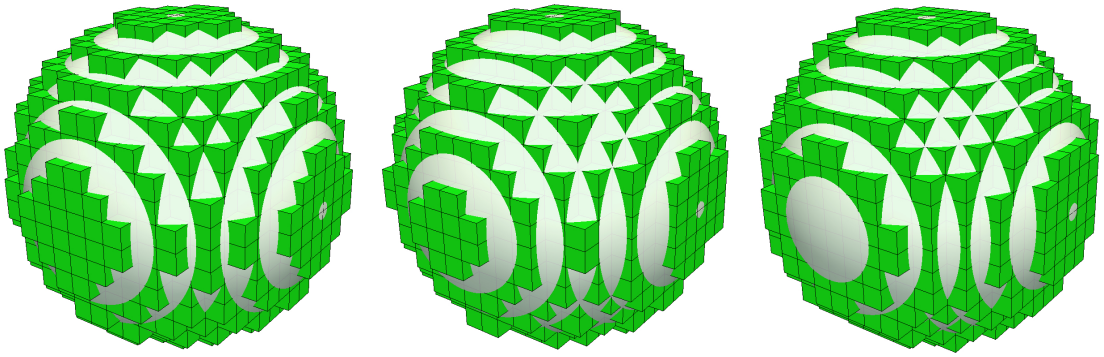
where $C < 1$ is a scaling constant similar to a CFL number and the Δx_i are the finite difference grid lengths in each direction, which, in principle can be different though they are taken equal in all examples this work. In practice, we find that using $C \approx 0.5$ tends to best balance numerical accuracy and computational efficiency.

Even though the flow solver by itself achieves second-order accuracy in time as verified in Chapter 5, it should be recognized that its combination with the particles is not fully second-order accurate. Indeed, as shown in 2.8, the force acting on the particle at time t^{n+1} is calculated using the newly-updated flow at t^{n+1} but maintaining the particle at the position that it occupied at time t^n. The same comment applies to the method reported in Z&P.

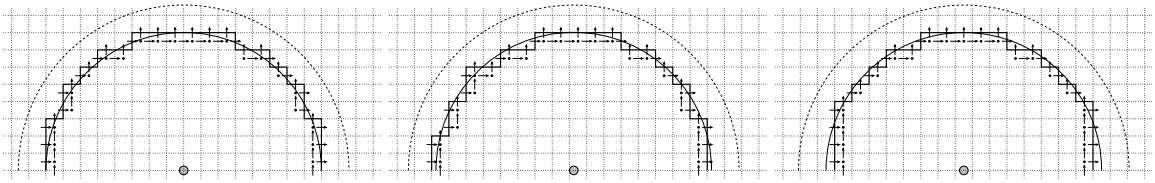
2.2 Cage construction

As a particle moves about the domain, its center may reside at an arbitrary location within a grid cell. The cage—the subset of discrete flow solver cells at which a particle's influence is communicated—adapts its shape to the position of the

particle, illustrated in Figure 2.1 for three particle positions: zero, $\frac{1}{4}$, and $\frac{1}{2}$ mesh lengths displaced from the cell corner. The motion of the particle causes the cage to change shape, with cells added to and removed from the cage at every time step. In reality, the cage shape changes even more smoothly than what is illustrated in the



(a) Three dimensional representation. The cage is a closed surface, part of which cannot be seen as it resides inside the particle.



(b) Two-dimensional planes through the centers of the Physalis cages in (a). Velocity and pressure boundary conditions are denoted by arrows and dots, respectively. The dashed line represents the integration surface used for the scalar products (2.21).

Figure 2.1: Three examples of the Physalis particle cage that specifies the locations of the analytic boundary conditions. The particles are positioned with center points at displacements 0, $\Delta x/4$, and $\Delta x/2$ from a cell corner.

three discrete positions in Figure 2.1 as particles typically move a small fraction of a cell in each time step.

We discuss the procedure for generating the particle cage in Section 4.3.1. In contrast to Z&P, this cage generation procedure is so fast that it permits the generation of cages for all particles at each time step, providing a very smooth transition between cage shapes as particles move with respect to the Cartesian grid. The no-slip condition satisfied at the particle surface guarantees that the center of a particle moves a fraction of a cell by adaptively determining the time step size using the fluid velocity field to maintain a specified CFL condition number as discussed in Section 2.1.2. Further, as a margin of safety, the time step size calculation takes into account the fluid cells inside the particle surface, which are set equal to the solid body velocity of the particle as explained in Section 4.3.4. As a consequence, at each time step, only a small number of cells are added to or removed from the cage, which minimizes the unavoidable force fluctuations affecting all fixed-grid methods. Further, the more smoothly transitioning cages assist Lamb's coefficient convergence.

2.3 Transformation to particle frame

The first step of the Physalis method proper—repeated independently for each particle in the flow—is to change our reference frame to that of a particle. We

introduce the fluid velocity $\mathbf{u}$ in the reference frame of a particle by writing

$$\mathbf{U} = \mathbf{u} + \mathbf{w} + \boldsymbol{\Omega} \times \mathbf{r}, \tag{2.9}$$

where $\mathbf{r} = (r, \theta, \varphi)$ is a position vector with respect to the center of the particle. Furthermore, we let

$$\mathbf{u} = \tilde{\mathbf{u}} + \frac{1}{10\nu}\left(\frac{r^5 - a^5}{r^3}\right)\dot{\boldsymbol{\Omega}} \times \mathbf{r}, \tag{2.10a}$$

$$p = \tilde{p} + \frac{1}{2}\rho_f \left(\boldsymbol{\Omega} \times \mathbf{r}\right)^2 + \rho_f \left(\mathbf{g} - \dot{\mathbf{w}}\right) \cdot \mathbf{r}, \tag{2.10b}$$

where the dots represent differentiation with respect to time. It can be shown (see Appendix A), with these definitions, that the Navier-Stokes equations (1.1) become

$$\frac{\partial \mathbf{u}}{\partial t} + \left(\mathbf{u} \cdot \nabla\right)\mathbf{u} + 2\boldsymbol{\Omega} \times \mathbf{u} = -\frac{1}{\rho_f}\nabla\tilde{p} + \nu\nabla^2\tilde{\mathbf{u}}, \tag{2.11}$$

$$\nabla \cdot \tilde{\mathbf{u}} = 0. \tag{2.12}$$

Note that, while the left-hand side of (2.11) is in terms of the fluid velocity $\mathbf{u}$ from the particle frame, the right-hand side is in terms of the modified fields $\tilde{\mathbf{u}}$ and $\tilde{p}$. Since, due to the no-slip condition, $\mathbf{u}$ vanishes at the particle surface, the left-hand side of (2.11), due to continuity, will be very small in the immediate neighborhood of the particle surface. Therefore, in a region near $r = a$, we may neglect the left-hand side of (2.11) and write

$$0 = -\nabla\tilde{p} + \mu\nabla^2\tilde{\mathbf{u}}. \tag{2.13}$$

Equations (2.12) and (2.13) are recognized as the Stokes equations. It is worth stressing that the region in which (2.13) is a good approximation to (2.11) decreases in size as particle Reynolds number increases, but that (2.13) is always a viable approximation—given sufficient resolution—for any finite Reynolds number. In this regard, the present method is no different from any other.

2.4 Lamb's solution for flow about a sphere

A solution to the Stokes equations (2.13) about a sphere with a no-slip surface, which we will use to pose boundary conditions to the flow solver, was given by Lamb (Lamb, 1932; Kim and Karilla, 1991) using spherical vector harmonics. We split the velocity and pressure fields into an incident external flow (superscript e) and a particle-induced perturbation flow (superscript p), namely,

$$\tilde{\mathbf{u}} = \mathbf{u}^e + \mathbf{u}^p, \tag{2.14a}$$

$$\tilde{p} = p^e + p^p. \tag{2.14b}$$

Lamb's general solution for (2.13) is (Lamb, 1932; Kim and Karilla, 1991)

$$\mathbf{u}^e(r,\theta,\varphi) = \frac{\nu}{a^2}\sum_{l=0}^{\infty}\frac{1}{(l+1)(2l+3)}\left[\frac{1}{2}(l+3)\,r^2\nabla p_l - l\mathbf{r}p_l\right] + \frac{\nu}{a}\sum_{l=0}^{\infty}\left[a\nabla\phi_l + \nabla\times(\mathbf{r}\chi_l)\right], \tag{2.15a}$$

$$\mathbf{u}^p(r,\theta,\varphi) = \frac{\nu}{a^2}\sum_{l=1}^{\infty}\frac{1}{l(2l-1)}\left[-\frac{1}{2}(l-1)\,r^2\nabla p_{-l-1} + (l+1)\,\mathbf{r}p_{-l-1}\right] + \frac{\nu}{a}\sum_{l=1}^{\infty}\left[a\nabla\phi_{-l-1} + \nabla\times(\mathbf{r}\chi_{-l-1})\right], \tag{2.15b}$$

$$p^e(r,\theta,\varphi) = \frac{\mu\nu}{a^2}\sum_{l=0}^{\infty}p_l, \tag{2.15c}$$

$$p^p(r,\theta,\varphi) = \frac{\mu\nu}{a^2}\sum_{l=0}^{\infty}p_{-l-1}, \tag{2.15d}$$

where the p_l, ϕ_l, and χ_l are solid harmonics of order l. For example,

$$\begin{aligned} p_l &= \left(\frac{r}{a}\right)^l\sum_{m=-l}^{l}p_{lm}Y_l^m(\theta,\varphi) \\ &= \left(\frac{r}{a}\right)^l\left[p_{l0}^{\Re}N_l^0P_l^0 + 2\sum_{m=1}^{l}N_l^mP_l^m\left(p_{lm}^{\Re}\cos m\varphi - p_{lm}^{\Im}\sin m\varphi\right)\right], \end{aligned} \tag{2.16}$$

where

$$Y_l^m(\theta,\varphi) = N_l^mP_l^me^{im\varphi}; \qquad -l\le m\le l, \tag{2.17}$$

with

$$N_l^m = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}}, \tag{2.18}$$

and associated Legendre functions P_l^m is a surface harmonic of order l. We refer the reader to Appendix B for more details about the spherical vector harmonics and their derivatives, and draw special attention to (B.9), which is a version of (2.15) that is ready to be implemented in code.

The $p_{lm} = p^{\Re}_{lm} + ip^{\Im}_{lm}$ are complex-valued nondimensional constants that we have broken into real and imaginary parts in the last step of (2.16). The solid harmonics ϕ_l and χ_l are written similarly in terms of coefficients ϕ_{lm} and χ_{lm}. The harmonics of negative order appearing in (2.15b) and (2.15d) can be related to the positive harmonics by using the no-slip condition to find

$$p_{-l-1} = -\frac{2l-1}{2\left(l+1\right)} l \left[p_l + 2\left(2l+1\right)\phi_l\right] \left(\frac{a}{r}\right)^{2l+1} \tag{2.19a}$$

$$\phi_{-l-1} = -\frac{a^2}{4}\frac{l}{l+1}\left[\frac{2l+1}{2l+3}p_l + 2\left(2l-1\right)\phi_l\right] \left(\frac{a}{r}\right)^{2l+1} \tag{2.19b}$$

$$\chi_{-l-1} = \left(\frac{a}{r}\right)^{2l+1} \chi_l. \tag{2.19c}$$

The expansions (2.15) converge spectrally. Truncations of the summations with $l_{\max} = 2,\ 3,$ and 4 requires a total of 25, 46, and 73 coefficients, respectively. We find that, in most situations up to $\mathrm{Re}_p \approx 150$ with particle resolution $a/\Delta x = 8$, the truncation order $l_{\max} = 2$ in the spherical harmonic expansion captures typical flow conditions well. Of course, increasing $\Re_p$ requires finer particle resolution and also larger $l_{\max}$. It may be noted that, for a given order l, there is not necessarily a hierarchy of coefficients of different m. All of the m are required to accommodate the various possible complex local flows that might occur over the course of a simulation.

It is worth noting that the Physalis method requires an analytic solution such as that given above by Lamb's solution for Stokes flow about a sphere, which effectively

limits the method to undeformable spherical particles only. There exists a similar (though significantly more treacherous) analytic solution for ellipsoids that has not been incorporated, but, without the availability of general analytic solutions, the basic idea of the method might be extended to particles of arbitrary shape by means of a boundary integral local solution of the Stokes equations. Such implementation unfortunately would not exploit one of the strong features of Physalis, which is the spectral convergence of Lamb's solution. Nevertheless, there are important reasons to extend the method beyond spheres, such as the study of the effect of particle anisotropy.

2.5 Determination of Lamb's coefficients

At every Lamb's coefficient iteration—the process by which the local and global flow solutions converge discussed in Section 2.6—the coefficients p_{lm}, ϕ_{lm}, and χ_{lm} must be determined from the currently available Navier-Stokes solution in order to determine the local analytic flow field. The various ways in which this objective can be achieved are discussed in G&P. In general, these calculations require the scalar product of spherical vector harmonics with the numerically determined $\tilde{\mathbf{u}}$ and $\tilde{p}$. The scalar products are defined by

$$(f,g) = \int_{-\pi}^{\pi} d\varphi \int_{0}^{\pi} \sin\theta \, d\theta \, \bar{f}\,(\theta,\varphi)\, g\,(\theta,\varphi)\,, \tag{2.20}$$

where the overbar denotes complex conjugation. It is concluded in G&P that the following three scalar products are optimal for determining Lamb's coefficients:

$$\frac{a^2}{\nu\mu}\left(Y_l^m,\tilde{p}\right)=\left[1-\frac{l\left(2l-1\right)}{2\left(l+1\right)}\left(\frac{a}{r}\right)^{2l+1}\right]\left(\frac{r}{a}\right)^{l}p_{lm} - \frac{l\left(2l-1\right)\left(2l+1\right)}{l+1}\left(\frac{a}{r}\right)^{l+1}\phi_{lm}, \tag{2.21a}$$

$$\frac{a}{\nu}\left(r\nabla Y_l^m,\tilde{\mathbf{u}}\right)=\frac{l}{4}\left\{\frac{2\left(l+3\right)}{2l+3}+\left[l-2-\frac{l\left(2l+1\right)}{2l+3}\left(\frac{a}{r}\right)^2\right]\left(\frac{a}{r}\right)^{2l+1}\right\}\left(\frac{r}{a}\right)^{l+1}p_{lm} + l\left\{l+1+\frac{1}{2}\left[\left(l-2\right)\left(2l+1\right)\left(\frac{r}{a}\right)^2-l\left(2l-1\right)\right]\left(\frac{a}{r}\right)^{2l+1}\right\}\left(\frac{r}{a}\right)^{l-1}\phi_{lm} \tag{2.21b}$$

$$\frac{a}{\nu}\left(\mathbf{r}\times\nabla Y_l^m,\tilde{\mathbf{u}}\right)=l\left(l+1\right)\left[\left(\frac{a}{r}\right)^{2l+1}-1\right]\left(\frac{r}{a}\right)^{l}\chi_{lm}. \tag{2.21c}$$

These are the scalar products that we have adopted in this work. More explicitly,

$$\begin{aligned}\left(Y_l^m,\tilde{p}\right)&=\left(Y_l^m,\tilde{p}\right)^{\Re}-i\left(Y_l^m,\tilde{p}\right)^{\Im}\\&=\int_{-\pi}^{\pi}d\varphi\int_0^{\pi}d\theta\,N_l^mP_l^m\tilde{p}\left(r,\theta,\varphi\right)\sin\theta\left[\cos\left(m\varphi\right)-i\sin\left(m\varphi\right)\right],\end{aligned} \tag{2.22a}$$

$$\begin{aligned}&\left(r\nabla Y_l^m,\tilde{\mathbf{u}}\right)^{\Re}\\&=\int_{-\pi}^{\pi}d\varphi\int_0^{\pi}d\theta\,N_l^m\left[\left(l-m+1\right)P_{l+1}^m-\left(l+1\right)\cos\theta P_l^m\right]\cos\left(m\varphi\right)\tilde{u}_\theta\left(r,\theta,\varphi\right)\\&-\int_{-\pi}^{\pi}d\varphi\int_0^{\pi}d\theta\,mN_l^mP_l^m\sin\left(m\varphi\right)\tilde{u}_\varphi\left(r,\theta,\varphi\right),\end{aligned} \tag{2.22b}$$

$$\begin{aligned}&\left(r\nabla Y_l^m,\tilde{\mathbf{u}}\right)^{\Im}\\&=-\int_{-\pi}^{\pi}d\varphi\int_0^{\pi}d\theta\,N_l^m\left[\left(l-m+1\right)P_{l+1}^m-\left(l+1\right)\cos\theta P_l^m\right]\sin\left(m\varphi\right)\tilde{u}_\theta\left(r,\theta,\varphi\right)\\&-\int_{-\pi}^{\pi}d\varphi\int_0^{\pi}d\theta\,mN_l^mP_l^m\cos\left(m\varphi\right)\tilde{u}_\varphi\left(r,\theta,\varphi\right),\end{aligned} \tag{2.22c}$$

$$
\begin{aligned}
&(\mathbf{r}\times\nabla Y_l^m,\tilde{\mathbf{u}})^{\Re}\\
&=\int_{-\pi}^{\pi}d\varphi\int_0^{\pi}d\theta\, mN_l^mP_l^m\sin(m\varphi)\,\tilde{u}_\theta(r,\theta,\varphi) \qquad (2.22d)\\
&+\int_{-\pi}^{\pi}d\varphi\int_0^{\pi}d\theta\, N_l^m\left[(l-m+1)P_{l+1}^m-(l+1)\cos\theta P_l^m\right]\cos(m\varphi)\,\tilde{u}_\varphi(r,\theta,\varphi),\\
&(\mathbf{r}\times\nabla Y_l^m,\tilde{\mathbf{u}})^{\Im}\\
&=\int_{-\pi}^{\pi}d\varphi\int_0^{\pi}d\theta\, mN_l^mP_l^m\cos(m\varphi)\,\tilde{u}_\theta(r,\theta,\varphi) \qquad (2.22e)\\
&-\int_{-\pi}^{\pi}d\varphi\int_0^{\pi}d\theta\, N_l^m\left[(l-m+1)P_{l+1}^m-(l+1)\cos\theta P_l^m\right]\sin(m\varphi)\,\tilde{u}_\varphi(r,\theta,\varphi).
\end{aligned}
$$

The effect of the choice of the radius r of the surface of integration has been studied in G&P, and on this basis we typically take $r/a\approx 1.2$ for the Reynolds numbers encountered in this study. We discuss in Section 4.3.3 the manner in which we evaluate the scalar products numerically.

Note that p_{lm} and ϕ_{lm} are coupled through (2.21a) and (2.21b). If we write these coupled equations as

$$
\begin{cases}
\dfrac{a^2}{\nu\mu}(Y_l^m,\tilde{p})=\mathcal{A}p_{lm}+\mathcal{B}\phi_{lm}\\[2ex]
\dfrac{a}{\nu}(r\nabla Y_l^m,\tilde{u})=\mathcal{C}p_{lm}+\mathcal{B}\phi_{lm}
\end{cases}
\qquad (2.23)
$$

for brevity, we conclude that

$$
p_{lm}=\frac{\frac{a^2}{\nu\mu}(Y_l^m,p)\,\mathcal{D}+\frac{a}{\nu}(r\nabla Y_l^m,\tilde{\mathbf{u}})\,\mathcal{B}}{\mathcal{A}\mathcal{D}+\mathcal{B}\mathcal{C}} \qquad (2.24a)
$$

$$
\phi_{lm}=\frac{\frac{a}{\nu}(r\nabla Y_l^m,\tilde{\mathbf{u}})\,\mathcal{A}-\frac{a^2}{\nu\mu}(Y_l^m,p)\,\mathcal{C}}{\mathcal{A}\mathcal{D}+\mathcal{B}\mathcal{C}} \qquad (2.24b)
$$

$$\chi_{lm} = \frac{\frac{a}{\nu}\left(\mathbf{y} \times \nabla Y_l^m, \tilde{\mathbf{u}}\right)}{l\left(l+1\right)\left[\left(\frac{a}{r}\right)^{2l+1} - 1\right]\left(\frac{r}{a}\right)^l}, \tag{2.24c}$$

where it is worth reiterating that Lamb's coefficients and the scalar products alike are complex valued.

2.6 Applying particle boundary conditions

We adapt the flow solver described in Section 2.1 to apply the particle boundary conditions determined by Lamb's solution at the locations specified by the particle cage. Because the process is iterative, at the κ-th step of Lamb's coefficient iterations, the finite-difference discretized Navier-Stokes equations become

$$\frac{\mathbf{U}_\kappa^* - \mathbf{U}^n}{\Delta t} = -\left[\left(\mathbf{U} \cdot \nabla_h\right)\mathbf{U}\right]_\kappa^{n+1/2} - \frac{1}{\rho_f}\nabla_h p^{n-1/2} + \nu\nabla_h^2 \mathbf{U}_\kappa^{n+1/2} + \mathbf{g}, \tag{2.25}$$

$$\mathbf{U}_{\kappa+1}^{n+1} = \mathbf{U}_\kappa^* - \frac{\Delta t}{\rho_f}\nabla_h \phi_\kappa^{n+1}, \tag{2.26}$$

$$p_\kappa^{n+1/2} = p^{n-1/2} + \phi_\kappa^{n+1}, \tag{2.27}$$

where the auxiliary field ϕ is determined from

$$\nabla^2 \phi_\kappa^{n+1} = \frac{\rho_f}{\Delta t}\nabla \cdot \mathbf{U}_\kappa^*. \tag{2.28}$$

We apply the velocity and pressure given by the Stokes solution at the cage cells. For a given set of Lamb's coefficients, the Stokes solution prescribes a specific flow

field on the integration surface used for the scalar products. However, the finite-difference solution obtained using the boundary conditions obtained from the same set of Lamb's coefficients will be incompatible with the Stokes solution unless the coefficients are correct. This mismatch drives the iterative process, which begins for new each time step with the converged coefficient values from the previous time step.

At each step of the iteration, we calculate the quantity

$$\frac{\left|\psi_{lm}^{\kappa} - \psi_{lm}^{\kappa-1}\right|}{|\psi_{lm}^{\kappa}|}, \tag{2.29}$$

where $\psi_{lm}^{\kappa} = \{p_{lm}^{\kappa}, \phi_{lm}^{\kappa}, \chi_{lm}^{\kappa}\}$ represents a generic Lamb's coefficient at iteration κ. The iterations are terminated when (2.29) become smaller than a prescribed tolerance, which we typically take as 10^{-2}. We have carried out extensive tests and concluded that using a more stringent tolerance has negligible effect on accuracy. To avoid unnecessary Lamb's coefficient iterations, (2.29) is calculated only for coefficients greater than 10^{-4} to 10^{-6} times the largest coefficient. In practice, the number of iterations required for convergence of Lamb's coefficients can vary from one to a few tens depending on many factors including the time step size, the amount of external forcing, the number of particles, and particle collisions. A typical number of sub-time-step iterations is approximately five. Upon convergence, we set the velocity and pressure nodes inside the particles to reflect the particle solid body motion as explained in Section 4.3.4. Unlike previous versions of Physalis (Z&P), the current method no longer suffers from the Lamb's coefficient convergence instability that demanded use of an underrelaxation scheme, although it still may help stabilize strong

particle collisions (see Chapter 3).

2.7 Hydrodynamic forces and couples

One of the biggest benefits of the Physalis method is that Lamb's coefficients determine the hydrodynamic forces, couples, etc. acting on the particles. Phrased another way, the process of determining the flow field also determines these forces, etc. without extra calculations. The hydrodynamic force $\mathbf{F}_h$ and couple $\mathbf{L}_h$ on the particle are expressed in terms of Lamb's coefficients by

$$\hat{\mathbf{F}}_h = \pi\mu\nu\left(2\boldsymbol{\Phi} - \mathbf{P}\right) + 2\pi\mu\nu\left(2\boldsymbol{\Phi} + \mathbf{P}\right) = \pi\mu\nu\left(6\boldsymbol{\Phi} + \mathbf{P}\right), \tag{2.30}$$

$$\hat{\mathbf{L}}_h = 8\pi\mu\nu a\boldsymbol{\chi}, \tag{2.31}$$

where $\mathbf{P} = -2N_1^1 p_{11}^{\Re}\hat{\mathbf{i}} + 2N_1^1 p_{11}^{\Im}\hat{\mathbf{j}} + N_1^0 p_{10}^{\Re}\hat{\mathbf{k}}$; $\boldsymbol{\Phi}$ and $\boldsymbol{\chi}$ are defined analogously. The separation into two terms of the force shown in (2.30) identifies the pressure and viscous contributions, respectively. These equations are written in the frame of the particle. The forces and couples in the laboratory frame of the calculation are obtained by adding the contribution of the transformation (2.10) with the result

$$\mathbf{F}_h = \rho_f v\left(\dot{\mathbf{w}} - \mathbf{g}\right) + \hat{\mathbf{F}}_h, \qquad \mathbf{L}_h = \rho_f v a^2\dot{\boldsymbol{\Omega}} + \hat{\mathbf{L}}_h, \tag{2.32}$$

with $v = \frac{4}{3}\pi a^3$ the particle volume.

2.8 Particle motion

The particle acceleration $\dot{\mathbf{w}}$ is computed by adding together the hydrodynamic force $\mathbf{F}_h$, externally imposed forces $\mathbf{F}_e$ (beyond gravity, if any), particle interaction forces $\mathbf{F}_i$ (see Chapter 3), and applied gravitational force per unit mass. The accelerations at the time $t^{n+1} = t^n + \Delta t$ and sub-time-step iteration number κ take the form

$$\dot{\mathbf{w}}_\kappa^{n+1} = \frac{1}{m}\left(\mathbf{F}_h^\kappa + \mathbf{F}_e^\kappa + \mathbf{F}_i^\kappa\right) + \mathbf{g}, \qquad \dot{\mathbf{\Omega}}_\kappa^{n+1} = \frac{1}{I}\left(\mathbf{L}_h^\kappa + \mathbf{L}_e^\kappa + \mathbf{L}_i^\kappa\right). \tag{2.33}$$

The counter κ appears in (2.33) because we update the particle linear and angular velocities at each iteration to improve particle interaction accuracy and stability. Now, the particle position $\mathbf{X}$ and velocity $\mathbf{w}$ are given by the trapezoidal rule as

$$\mathbf{w}_\kappa^{n+1} = \mathbf{w}^n + \frac{1}{2}\left(\dot{\mathbf{w}}_\kappa^{n+1} + \dot{\mathbf{w}}^n\right)\Delta t, \qquad \mathbf{X}^{n+1} = \mathbf{X}^n + \frac{1}{2}\left(\mathbf{w}^{n+1} + \mathbf{w}^n\right)\Delta t. \tag{2.34}$$

The angular velocity is updated similarly:

$$\mathbf{\Omega}_\kappa^{n+1} = \mathbf{\Omega}^n + \frac{1}{2}\left(\dot{\mathbf{\Omega}}_\kappa^{n+1} + \dot{\mathbf{\Omega}}^n\right)\Delta t. \tag{2.35}$$

Even though the spherical symmetry decouples the particle orientation from the flow solver, we still track it by attaching an orthonormal basis $\{\mathbf{a}_x,\ \mathbf{a}_y,\ \mathbf{a}_z\}$ to the particle and integrating its motion using $\mathbf{\Omega}$ and quaternion rotations.

The equations change slightly if the body force $\mathbf{g}$ in the fluid momentum equation is incorporated in a modified pressure. In this case, the first of (2.32) becomes

$$\mathbf{F}_h = \rho_f v \dot{\mathbf{w}} + \hat{\mathbf{F}}_h, \tag{2.36}$$

and the first of (2.33) becomes

$$\dot{\mathbf{w}}_\kappa^{n+1} = \frac{1}{m}\left(\mathbf{F}_h^\kappa + \mathbf{F}_e^\kappa + \mathbf{F}_i^\kappa\right) + \frac{\rho_p - \rho_f}{\rho_p}\mathbf{g}. \tag{2.37}$$

It is well known that the added mass effects have an adverse influence on the stability of explicit methods for particles lighter than the fluid (Prosperetti and Tryggvason, 2009). It can be deduced from Equation (5.14) of the stated reference that an explicit method becomes unstable for $\rho_f/\rho_p < C_{am}$ where C_{am} is the added mass coefficient (equal to one half for an isolated sphere in uniform potential flow). We have verified that this stability limit applies to the present algorithm in spite of its partial implicitness. We conclude, therefore, that the partially implicit character of our method for particle integration is insufficient to stabilize the particle advancement. We retain this procedure because it proves more stable in the presence of collisions.

Chapter 3

Collision model

As particles approach each other, the number of fluid cells separating them decreases until contact. For collisions in viscous fluids—between particles or between a particle and a wall—we draw attention to two phases of a collision. During the first phase, when particles are near to each other but not yet touching, viscous lubrication forces dominate the interaction. Since the spatial step remains constant, we are unable to resolve the flow between the particles as they come near and must therefore model the missing viscous forces. The lubrication phase has been rather well investigated, and has been adopted in many methods as an important part of any collision model. We review the lubrication model in Section 3.1. We refer to the second phase—the time in which particles are touching one another—as the contact phase. Once contact has been made, we adopt a second model for repulsively separating the particles. Drawing from the granular flow literature where the influence of

the interstitial fluid is neglected, we find a rich history of particle contact modeling, which is well summarized in Crowe et al. (2012).

Particle-wall and particle-particle collisions in viscous fluids have been considered both experimentally and computationally for some time. Many experimental results published in the literature involve the normally-directed particle-wall collisions resulting from spherical particles falling under gravity immersed in a fluid, including those of McLaughlin (1968), ten Cate et al. (2002) and Gondret et al. (2002). Other results, such as those of Joseph et al. (2001) investigate the normal particle-wall collision of a spherical particle swinging as a pendulum into a wall, and Joseph and Hunt (2004) extends these experiments to oblique collisions. All of these experiments find, in general, the relation proposed by Davis et al. (1986): The coefficient of restitution

$$e = -\frac{w_1}{w_0}, \tag{3.1}$$

with w_0 and w_1 the particle velocity before and after the collision, respectively, depends strongly on the Stokes number of the collision,

$$\mathrm{St} = \frac{1}{9}\frac{\rho_p}{\rho_f}\mathrm{Re_p}, \tag{3.2}$$

where $\mathrm{Re}_p = 2aw_0/\nu$ is the Reynolds number based on the particle diameter $2a$. Here, the Stokes number may be interpreted as the ratio of the particle inertia to the viscous force.

In order to further motivate the present work, we summarize the contact models adopted in some recent resolved-particle simulations. All of these methods take a

"soft sphere" approach, which essentially poses an elastic repulsive force dependent on the amount that the particles overlap. (We should note that, although the particles overlap slightly in the soft sphere model, their material deformations are not resolved during the contact.) Li et al. (2012) uses a modified Hertzian contact force, which is, briefly, a nonlinear elastic response. Simeonov and Calantoni (2012) applies a linear elastic response that attempts to account "for plastic losses as the work during loading exceeds the work during unloading" by limiting the applied force by a factor related to the maximum overlap. Kempe and Fröhlich (2012b) fixes the number of time steps over which to smooth the contact in order to determine stiffness and damping coefficients for a damped Hertzian contact force. Finally, Kidanemariam and Uhlmann (2014) uses a damped linear elastic response. Of these works, Kempe and Fröhlich (2012b) provides the most generally applicable model, as it most accurately takes into account the actual material properties of the particles. The others inevitably make some arbitrary assumptions about the form of the elastic response or, worse yet, the values of the stiffness and damping coefficients.

The present work proposes a contact model, discussed in Section 3.2, that adopts a nonlinearly damped Hertzian response. The model relies upon real material properties (except for a softened Young's modulus) and flow conditions to determine the damping coefficient in order to produce the expected coefficient of restitution. We tune the model to match the experimental results discussed above using a single input parameter. Section 3.3 discusses our method of accounting for cases when a particle

contacts more than one other particle at a time and Section 5.4 documents extensive validation of the model against various experiments in the literature. The majority of this chapter has been adapted from Sierakowski (2016b).

3.1 Lubrication

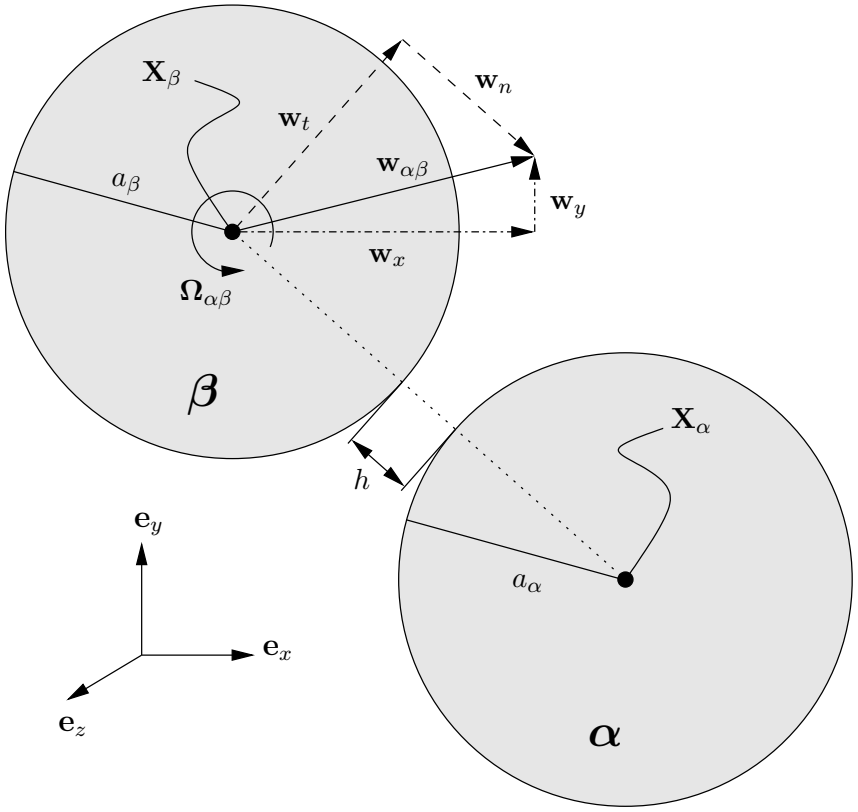


Figure 3.1: The relative motion of particle β from particle α in the normal/tangential reference frame.

Consider the interaction of two particles α and β with positions $\mathbf{X}_\alpha$ and $\mathbf{X}_\beta$. We

decompose the relative motion between the two particles into normal and tangential components for each interacting pair. We define the unit normal as

$$\hat{\mathbf{n}} = \frac{\mathbf{X}_\beta - \mathbf{X}_\alpha}{|\mathbf{X}_\alpha - \mathbf{X}_\beta|}, \tag{3.3}$$

and from that determine the normal and tangential components of the relative particle velocity

$$\mathbf{w}_{\alpha\beta} = \mathbf{w}_\alpha - \mathbf{w}_\beta, \tag{3.4}$$

namely,

$$\mathbf{w}_n = \mathbf{w}_{\alpha\beta} \cdot \hat{\mathbf{n}}, \tag{3.5a}$$

$$\mathbf{w}_t = \mathbf{w}_{\alpha\beta} - w_n \hat{\mathbf{n}}. \tag{3.5b}$$

so that the tangential unit vector is

$$\hat{\mathbf{t}} = \frac{\mathbf{w}_t}{|\mathbf{w}_t|} \tag{3.6a}$$

and the binormal is

$$\hat{\mathbf{b}} = \hat{\mathbf{n}} \times \hat{\mathbf{t}}. \tag{3.6b}$$

In the event that $\mathbf{w}_t$ is zero, we define the reference frame in the opposite order, first using the relative angular velocity

$$\mathbf{\Omega}_{\alpha\beta} = \mathbf{\Omega}_\alpha + \mathbf{\Omega}_\beta, \tag{3.7}$$

to specify the binormal component

$$\hat{\mathbf{b}} = \frac{\mathbf{\Omega}_{\alpha\beta}}{|\mathbf{\Omega}_{\alpha\beta}|} \tag{3.8a}$$

before completing the basis using

$$\hat{\mathbf{t}} = \hat{\mathbf{b}} \times \hat{\mathbf{n}}. \tag{3.8b}$$

If both $\mathbf{w}_t$ and $\boldsymbol{\Omega}_{\alpha\beta}$ are zero, then we simply choose

$$\hat{\mathbf{t}} = \hat{\mathbf{e}}_x \tag{3.9}$$

and complete the basis using (3.6b). Figure 3.1 illustrates this reference frame.

As two particles α and β approach each other, the distance between them

$$h = |\mathbf{X}_\alpha - \mathbf{X}_\beta| - a_\alpha - a_\beta \tag{3.10}$$

decreases. Long-distance particle hydrodynamic interactions—when $h > a_\alpha$, for example—are resolved by the flow solver. Since the spatial step size remains constant, we are unable to resolve the flow between the particles as h approaches 0, and therefore supplement the hydrodynamic force of particle β on particle α with the following model that incorporates the effect of lubrication (Simeonov and Calantoni,

2012; Kim and Karilla, 1991; Jeffrey, 1982; Sierakowski and Prosperetti, 2016):

$$\mathbf{F}_n = 6\pi\mu a_\alpha \left[\frac{a_{\beta\alpha}^2}{\left(1+a_{\beta\alpha}\right)^2} \left(\frac{a_\alpha}{h} - \frac{a_\alpha}{\varepsilon} \right) + \frac{a_{\beta\alpha}\left(1+7a_{\beta\alpha}+a_{\beta\alpha}^2\right)}{5\left(1+a_{\beta\alpha}\right)^3} \ln\left(\frac{\varepsilon}{h}\right) \right] \left(\mathbf{w}_{\alpha\beta}\cdot\hat{\mathbf{n}}\right)\hat{\mathbf{n}}, \tag{3.11a}$$

$$\mathbf{F}_t = 6\pi\mu a_\alpha \left[\frac{4a_{\beta\alpha}\left(2+a_{\beta\alpha}+2a_{\beta\alpha}^2\right)}{15\left(1+a_{\beta\alpha}\right)^3} \left[\mathbf{w}_{\alpha\beta} - \left(\mathbf{w}_{\alpha\beta}\cdot\hat{\mathbf{n}}\right)\hat{\mathbf{n}}\right] - \frac{2a_{\beta\alpha}^2}{15\left(1+a_{\beta\alpha}\right)^2} \left(\boldsymbol{\Omega}_{\alpha\beta} + 4a_{\beta\alpha}^{-1}\boldsymbol{\Omega}_\alpha + 4a_{\beta\alpha}\boldsymbol{\Omega}_\beta\right)\times\hat{\mathbf{n}} \right] \ln\left(\frac{\varepsilon}{h}\right), \tag{3.11b}$$

$$\mathbf{L}_b = -8\pi\mu a_\alpha^2 \left\{ a_\alpha \frac{2a_{\beta\alpha}}{5\left(1+a_{\beta\alpha}\right)} \left[\left(\boldsymbol{\Omega}_\alpha + \frac{1}{4}a_{\beta\alpha}\boldsymbol{\Omega}_\beta\right) - \left(\boldsymbol{\Omega}_\alpha + \frac{1}{4}a_{\beta\alpha}\boldsymbol{\Omega}_\beta\right)\cdot\hat{\mathbf{n}}\hat{\mathbf{n}} \right] - \frac{a_{\beta\alpha}\left(4+a_{\beta\alpha}\right)}{10\left(1+a_{\beta\alpha}\right)^2}\mathbf{w}_{\alpha\beta}\times\hat{\mathbf{n}} \right\} \ln\left(\frac{\varepsilon}{h}\right), \tag{3.11c}$$

where $a_{\beta\alpha} = a_\beta/a_\alpha$ and $\varepsilon = a_\alpha$ is a parameter that enforces compact support of the model. This accounts for the first phase of a collision described above. We extend the lubrication model to particle-wall contact by setting $a_\beta \to \infty$ in (3.11).

3.2 Nonlinearly damped Hertzian contact

For the second phase of the collision—that is, when $h < 0$—we have contact and propose the nonlinearly damped Hertzian contact model discussed presently. To compute the normal component of the repulsive force on particle α in contact with particle β, we begin by considering the work of Tsuji et al. (1992), which applies the damped Hertzian repulsive force

$$\mathbf{F}_n = \left[-k_n\left(-h\right)^{3/2} - \eta_n\mathbf{w}_{\alpha\beta}\cdot\hat{\mathbf{n}}\right]\hat{\mathbf{n}}, \tag{3.12}$$

with spring constant k_n and damping constant η_n. We choose the shorthand notation

$$w = \mathbf{w}_{\alpha\beta} \cdot \hat{\mathbf{n}} \tag{3.13}$$

for the remainder of this section. For Hertzian contact, the spring constant takes the form

$$k_n = \frac{4}{3}\left(\frac{1-\sigma_\alpha^2}{E_\alpha} + \frac{1-\sigma_\beta^2}{E_\beta}\right)^{-1}\left(\frac{a_\alpha a_\beta}{a_\alpha + a_\beta}\right)^{1/2}, \tag{3.14}$$

where E_α and σ_α are the Young's modulus and Poisson's ratio of particle α, respectively. We define properties for particle β analogously. Researchers typically choose a constant value for damping coefficient η_n, which, as we will see below, corresponds to a specific coefficient of restitution. The present discussion relies on a method for determining the η_n that produces coefficients of restitution that align with values expected from experimental results published in the literature.

We attribute the following development to Tsuji et al. (1992), which considers the equation of motion associated with the proposed form of the repulsive force (3.12) under the change of variable $x = -h$, namely,

$$m_{\alpha\beta}\frac{d^2x}{dt^2} + \eta_n\frac{dx}{dt} + k_n x^{3/2} = 0, \tag{3.15}$$

with reduced mass $m_{\alpha\beta} = m_\alpha m_\beta / (m_\alpha + m_\beta)$. The dynamical system (3.15) describes the relative motion of the two particles depending on their material properties as they overlap with length $x = -h$. As the coefficient of restitution e varies only with Stokes number (Davis et al., 1986), it must be independent of the combination of parameters

$m_{\alpha\beta}$, η_n, and k_n. By proposing η_n take the form

$$\eta_n = \zeta \left(m_{\alpha\beta} k_n\right)^{1/2} x^{1/4}, \tag{3.16}$$

with free damping parameter ζ and rewriting (3.15) using (3.16) and the velocity $w = dx/dt$ to eliminate time, we find

$$w\frac{dw}{dx} + \zeta \left(\frac{k_n}{m_{\alpha\beta}}\right)^{1/2} x^{1/4} w + \frac{k_n}{m_{\alpha\beta}} x^{3/2} = 0. \tag{3.17}$$

Choosing the nondimensionalization

$$\hat{x} = x \left(m_{\alpha\beta}/k_n w_0^2\right)^{-2/5}; \qquad \hat{w} = \frac{w}{w_0}, \tag{3.18}$$

we rewrite (3.17) as

$$\hat{w}\frac{d\hat{w}}{d\hat{x}} + \zeta \hat{x}^{1/4} \hat{w} + \hat{x}^{3/2} = 0, \tag{3.19}$$

which is a nonlinear differential equation relating velocity to only the free parameter ζ. By integrating (3.19) from $\hat{x} = 0$ through the contact (that is, $x > 0$) and back to $\hat{x} = 0$, we find the response of the contact.

Figure 3.2 plots the solution of (3.19) for various values of ζ. Beginning with $\hat{h} = -\hat{x} = 0$ and $\hat{w} = -1$, we numerically integrate (3.19) through the contact until $\hat{h}$ returns to zero. When $\hat{h}$ has returned to zero, we see that varying ζ, which effectively varies η_n as expressed in (3.16), results in different values of $\hat{w}$ and, consequently, different values of the coefficient of restitution $e = -w_1/w_0 = \hat{w}|_{\hat{h}=0}$.

Figure 3.3 plots the relationship between the velocity at the end of the contact—namely, $e = \hat{w}|_{\hat{h}=0}$, the coefficient of restitution—and ζ. As (3.19) is nonlinear and

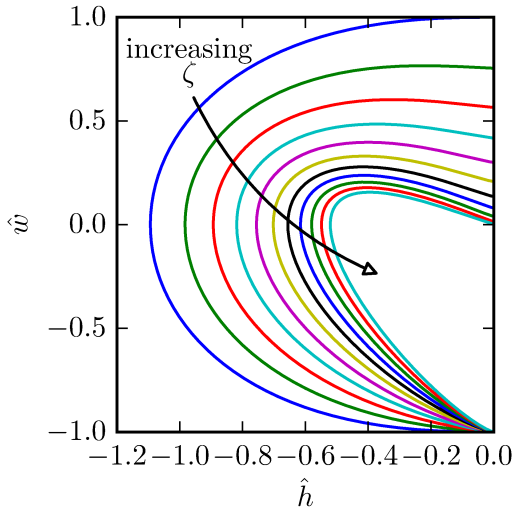


Figure 3.2: Solutions to (3.19) with various ζ. Contact begins with $\hat{w} = -1$ at $\hat{h} = 0$ and proceeds until $\hat{h} = 0$ again. The value of $\hat{w}$ at the end of contact is equal to e.

relatively time-consuming to solve, we fit the aforementioned curve with the following power law relation:

$$\zeta = 2.22 - 2.26e^{0.395}. \tag{3.20}$$

In general, we see that $0 \leq \zeta \leq 2$ as $0 \leq e \leq 1$ with larger values of ζ, which increases the damping coefficient η_n from (3.16), corresponding to smaller values of e. If we can predict the coefficient of restitution that corresponds to the Stokes number characterizing a collision, we can appropriately select ζ using (3.20) to provide the damping coefficient η_n that results in the accurate coefficient of restitution e.

The work of Barnocky and Davis (1988), summarized in Joseph et al. (2001), provides for us a sufficient prediction for the coefficient of restitution e expected for

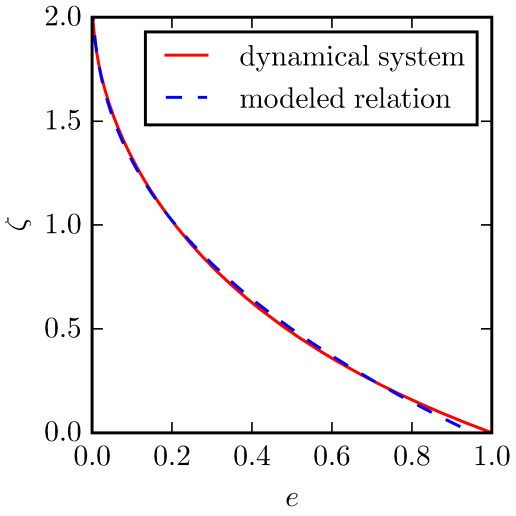


Figure 3.3: A plot of the relation (3.20) that models the coefficient of restitution $e = \hat{w}$ from many solutions of (3.19) with various ζ.

a given Stokes number. Consider the leading term of the lubrication force (3.11a) exerted on particle α as it approaches particle β, namely,

$$F_n = 6\pi\mu w a \left(\frac{a}{h}\right) = -m_\alpha \frac{dw}{dt} = -m_\alpha w \frac{dw}{dx}. \tag{3.21}$$

We may estimate the viscous dissipation of the collision by first integrating the approach from some h_0 to the critical distance h_c where (3.21) breaks down due to the particle's surface roughness. Assuming contact begins with velocity w_0 at h_0, we find that (3.21) generates the ratio

$$\int_{h_0}^{h_c} 6\pi\mu a^2 \frac{dx}{x} = \int_{w_0}^{w_c} -m\, dw \;\Rightarrow\; \frac{w_c}{w_0} = 1 + \frac{1}{\mathrm{St}_0} \log \frac{h_c}{h_0}, \tag{3.22}$$

where St_0 is the Stokes number at the onset of contact. Assuming that the velocity

after contact is related to the velocity before contact only through the dry coefficient of restitution e_{dry}—which is a material property of the pair of bodies in contact—we integrate (3.21) back to h_0 beginning with rebound velocity $e_{\text{dry}} w_c$ at h_c. We find

$$\int_{h_c}^{h_0} 6\pi\mu a^2 \frac{dx}{x} = \int_{e_{\text{dry}} w_c}^{w_1} -m\, dw \;\Rightarrow\; \frac{w_1}{w_0} = e = e_{\text{dry}} \frac{w_c}{w_0} + \frac{1}{\text{St}_0} \log \frac{h_c}{h_0}. \tag{3.23}$$

Combining (3.22) and (3.23) leaves us with a model for the relationship between St_0 and e, namely,

$$e = e_{\text{dry}} + \frac{1 + e_{\text{dry}}}{\text{St}_0} \log \frac{h_c}{h_0}, \tag{3.24}$$

where h_c/h_0 is, for our purposes, a tuning parameter to use to match experimental results. Joseph et al. (2001) found that choosing $h_c/h_0 = 10^{-3}$ fits the experimental results in the literature, but we adopt $h_c/h_0 = 10^{-4}$ in the present work since it generates a simulated response better aligned with experimental expectations, as is shown in Section 5.4. Operationally, we enforce that (3.24) produce a value of e between 0 and 1. As e_{dry} is a property of the pair of interacting materials, we assume that e_{dry} for a given material is that of a material contacting itself. If the particles are of different material, we take e_{dry} to be the average of the respective e_{dry} for each particle. We extend this model to particle-wall contact by setting $a_\beta \to \infty$ in (3.14).

We summarize our proposed contact model as follows:

1. At the onset of contact between two particles, determine the Stokes number St_0 according to (3.2);

2. Use (3.24) to estimate the anticipated coefficient of restitution of the contact;

3. Determine the corresponding damping parameter ζ with (3.20);

4. Apply the damping coefficient η_n given by (3.16) in the Hertzian contact force (3.12) for the duration of the contact.

Note that this method requires the storage of St_0 for the duration of the contact for each pair of contacting particles. We discuss the algorithm we have developed for maintaining a list of St_0 corresponding to each contacting pair in Section 3.3.

The only input to the model that fails to precisely match that of the true system is that of Young's modulus E. Indeed, the time scales associated with the true values of E for stiff materials such as glass and metal are orders of magnitude smaller than the time scales over which the fluid system evolves (Kempe and Fröhlich, 2012b). In the same way as all other soft sphere models, we wish, first of all, to avoid such time step restrictions, and, second, to avoid numerical instabilities associated with such a stiff system. We therefore soften E in our model by several orders of magnitude. This solution has been verified previously, and Joseph et al. (2001) specifically notes that the value of E has small effect on the relationship between coefficient of restitution and Stokes number. It has been recognized that softening E increases the time over which the contact occurs (Kempe and Fröhlich, 2012b), but we illustrate in Section 5.4 that this does not appear to decrease the accuracy of our model. As a rule of thumb for choosing the value of E, we make it as stiff as possible to prevent excessive overlap that would correspond to significant particle deformation (overlap of a few percent of the particle radius is our limit), yet still soft enough that the simulation

does not fail to converge to a solution.

For the tangential component of the contact force, we refer to the work of Simeonov and Calantoni (2012), which assumes that two particles in contact either stick (i.e., roll) or slide, depending on a coefficient of friction μ_f. We begin by calculating the relative tangential velocity at the contact point

$$\frac{d\mathbf{s}}{dt} = \mathbf{w}_c - (\mathbf{w}_c \cdot \hat{\mathbf{n}})\,\hat{\mathbf{n}}, \tag{3.25}$$

written here as the rate of change of the tangential slip at the contact point $\mathbf{s}$, where

$$\mathbf{w}_c = \mathbf{w}_{\alpha\beta} - \frac{1}{2}\,(a_\alpha + a_\beta + h)\,\boldsymbol{\Omega}_{\alpha\beta} \times \hat{\mathbf{n}} \tag{3.26}$$

accounts for both relative linear and angular motion. We next calculate the elastic tangential force by integrating

$$\frac{d\mathbf{F}_t}{dt} = k_t \frac{d\mathbf{s}}{dt} \tag{3.27}$$

with stiffness given by Mindlin's theory (Crowe et al., 2012)

$$k_t = 8\left(\frac{1-\sigma_\alpha^2}{H_\alpha} + \frac{1-\sigma_\beta^2}{H_\beta}\right)^{-1}\left(\frac{a_\alpha a_\beta}{a_\alpha + a_\beta}\right)^{1/2}(-h)^{1/2}; \qquad H = \frac{E}{2\,(1+\sigma)}, \tag{3.28}$$

and $\mathbf{F}_t = 0$ at the beginning of the contact. To ensure that the tangential force coincides with the tangential plane—which is itself moving in general—we project the resulting force back onto the current tangential plane:

$$\mathbf{F}_t = \mathbf{F}_t - (\mathbf{F}_t \cdot \hat{\mathbf{n}})\,\hat{\mathbf{n}}. \tag{3.29}$$

Should the force (3.29) exceed that which can be supported by frictional coefficient μ_f,

$$|\mathbf{F}_t| \leq \mu_f\,|\mathbf{F}_n|\,, \tag{3.30}$$

it is set equal to the maximum sustainable value, namely,

$$\mathbf{F}_t = \mu_f \left|\mathbf{F}_n\right| \frac{\mathbf{F}_t}{\left|\mathbf{F}_t\right|}. \tag{3.31}$$

Finally, the tangential motion induces the following couple:

$$\mathbf{L}_b = -\left(a_\alpha + \frac{1}{2}h\right)\left(\mathbf{F}_n + \mathbf{F}_t\right) \times \hat{\mathbf{n}}. \tag{3.32}$$

The sum of these forces and moments appear in (2.33) as interaction forces and moments $\mathbf{F}_i$ and $\mathbf{L}_i$, respectively.

3.3 Multiple contact

In a simulation with n_p particles, any given particle may simultaneously contact more than one other particle at a time. In this section, we the revisit the models introduced above—which describe contact between only two particles—and discuss the methods by which the models may be applied to simultaneous contact of multiple particles.

In general, we apply an iterative approach to reconciling the various forces applied by contact with multiple particles. During each iteration, we apply the binary interaction models introduced above and update the velocity and acceleration of every particle after each iteration according to the forces resulting from the interaction models and flow solver together. This iteration process is especially important since each of the binary interaction models relies on the relative velocity of the particles.

Any method of simulating disperse particle flows must include this iteration process for every time step in order to ensure that the multiple particle system arrives at a self-consistent state before proceeding to the next time step. Since the Physalis method that we employ for simulating the particle flow already requires multiple iterations per time step to match the local solution for each particle to the global solution of the flow solver, we combine the two iteration processes to limit the total amount of work. Note that, though the interaction models depend on relative particle position, we do not update this quantity during the iteration process because it introduces too much variability to the Physalis iteration process and causes the solution to diverge.

In addition to the iterative method, we recall that the damped Hertzian contact model requires an estimate of St_0, the Stokes number just before contact, for every contact. Further, this quantity must be retained for the duration of the contact as it is used to estimate the coefficient of restitution (3.24) that determines the damping parameter ζ (3.20), which in turn sets the damping coefficient η (3.16). As such, we store a list of all contacting particles and their corresponding St_0 for the duration of the contact. To avoid allocating and freeing memory when recording a new contacting pair, we begin with a list long enough to contain the maximum number of particles that can contact at any time. According to the theory of hexagonal close packed or face-centered cubic arrangements (see Figure 3.4)—which are known to maximize volume fraction of spheres—any particle may only be in contact with 12 other particles of the same size at once. (Note that this number may change significantly if

simulating particles of different size.) We therefore carry two lists of length 12 for each particle: the first list stores the number by which we reference the other particles in contact (between zero and $n_p - 1$) and the second list stores St_0 corresponding to that contacting pair. We initialize the list of particle references to -1, which indicates there are no particles contacting this particle. For each time step involving a contact, we perform these steps for the contacting particles:

1. Check whether the contacting particle is already in the particle reference list;
2. If yes, continue the time step using the damped Hertzian contact model with ζ determined from the stored St_0 corresponding to the contacting particle;
3. If no:
   (a) Find the first element in the list of particle references equal to -1;
   (b) Set this element equal to the reference number of the contacting particle;
   (c) Compute St_0 (3.2) for the particle pair and store it in the corresponding element of the list of contacting Stokes numbers;
4. Check whether the particles are still in contact at the end of the time step;
5. If no: set the reference to the particle that is no longer in contact equal to -1.

As an example of the ability of our methods to seamlessly account for multiple particle contact, we propose the following testing simulation. We arrange thirteen particles of radius $a = 0.5$ in a face-centered cubic lattice as illustrated in Figure

3.4, with one at the center at $(0, 0, 0)$ and the remaining twelve at the vertices of a cuboctahedron of side length $6a/\sqrt{2}$, namely, $6a\,(0, \pm 1, \pm 1)$, $6a\,(\pm 1, 0, \pm 1)$, and $6a\,(\pm 1, \pm 1, 0)$. In a normalized fluid with $\rho_f = \nu = 1$, we accelerate each of the cuboctahedral particles towards the center using a linear spring with a stiffness coefficient of $K = 5,000$ and a relaxed length of $2a$ so there is no applied force at the moment of contact with the center particle. The triply-periodic domain is of size $(8a)^3$ and resolution 128^3. Each particle has the following material properties: $\rho_p = 10\rho_f$, $E = 5 \times 10^7$, $\sigma = 0.5$, $e_{\text{dry}} = 1$, and $\mu_f = 0.5$.

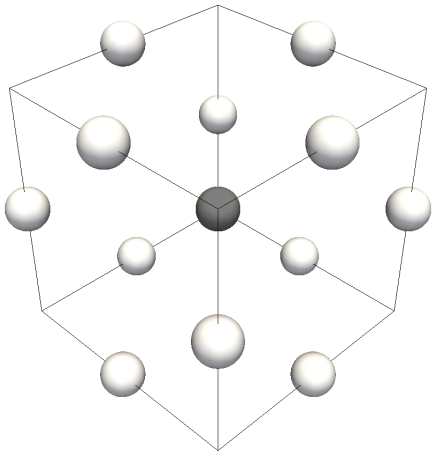
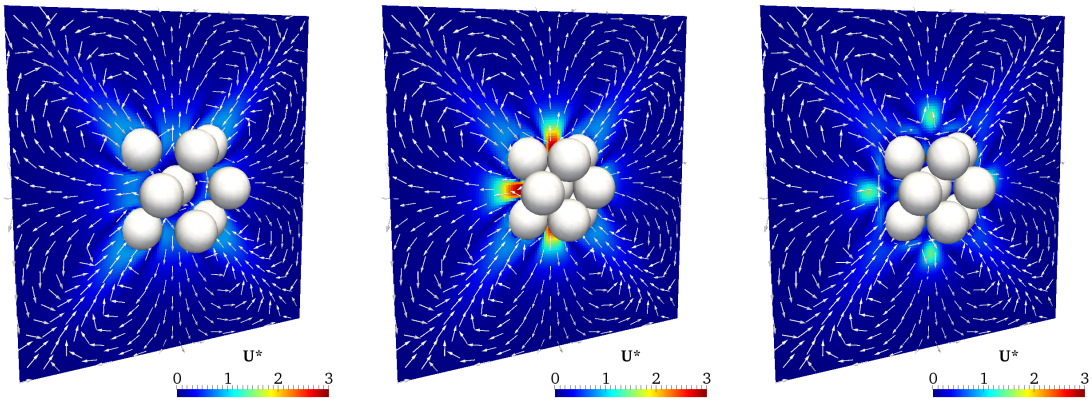


Figure 3.4: The initial positions of the particles in a face-centered cubic lattice. The darker particle sits at the center of the lattice. The connecting lines do not indicate the extent of the domain; it is larger.

We adopt

$$t_K = \sqrt{\frac{\frac{4}{3}\pi a^3 \rho_p}{K}} \tag{3.33}$$



(a) $t/t_K = -0.15$ (b) $t/t_K = 0$ (c) $t/t_K = 0.15$

Figure 3.5: The cuboctahedral collision flow field before, during, and after contact, where $\mathbf{U}^* = |\mathbf{U}| \,/\, (h_0/t_K)$.

as the time scale driving the particle motion and render visualizations of the flow field at times $t/t_K = \{-0.15, 0, 0.15\}$ in Figure 3.5, where $t/t_K = 0$ corresponds to the onset of contact. During their approach, shown in Figure 3.5a, the cuboctahedral particles entrain fluid to form wakes, but they also displace fluid as they progress. The fluid being displaced is clearly evident at contact, shown in Figure 3.5b, as distinct jets develop between the particles in order for the fluid trapped between the cuboctahedral particles and the center particle to escape. After contact, this pulse of fluid momentum can clearly be seen propagating away from the center particle, as is shown in Figure 3.5c.

Figure 3.6 plots the normalized separation length h/a, normal velocity $w_n/\left(h_0/t_K\right)$, and normal force $F_n/F_{n,0}$ on the cuboctahedral particles for the duration of the

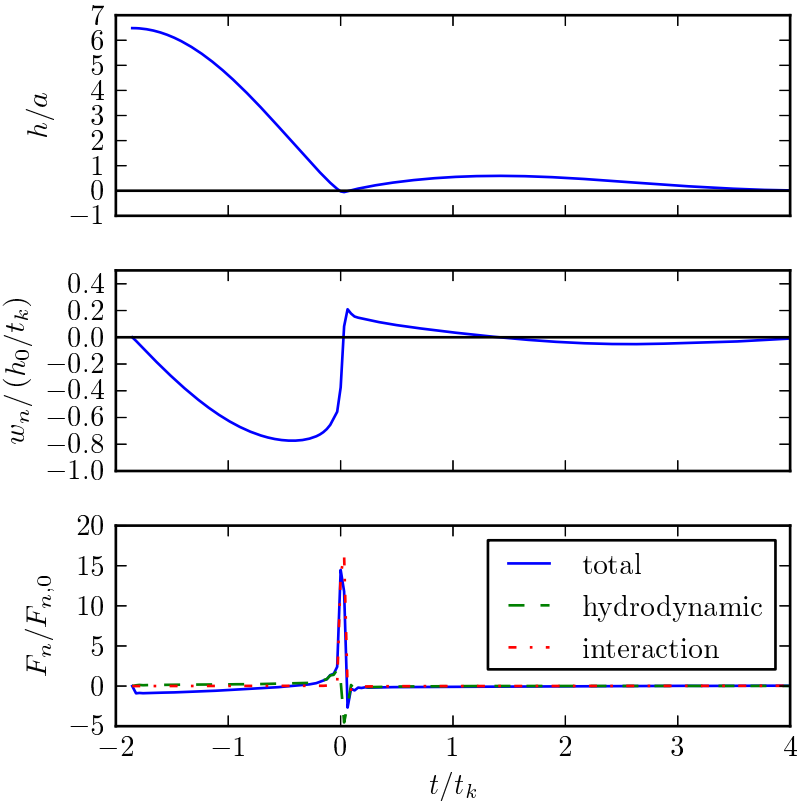


Figure 3.6: The separation length h/a, normal velocity $w_n/\left(h_0/t_K\right)$, and normal force $F_n/F_{n,0}$ on the cuboctahedral particles for the duration of the simulation.

simulation, where h_0 is the initial separation length and $F_{n,0} = kh_0$ is the initial spring force on the particles. Due to symmetry, contact occurs simultaneously with $\mathrm{St} = 41.4$ and a corresponding coefficient of restitution $e = 0.571$ for all particles; we have plotted this result along with other particle-particle contact validation tests in Figure 5.8. Note the strength of the hydrodynamic interactions between the particles just before contact when the jets seen in Figure 3.5 are strongest, indicating significant viscous dissipation of energy. Interestingly, these jets move with a velocity more than three times that of the maximum particle velocity. We see that the second contact—after the initial rebound—approaches with too small a Stokes number to rebound again. Since symmetry is maintained throughout the simulation and all particles rebound with an appropriate coefficient of restitution, this test indicates that the multiple particle contact is operating properly.

Chapter 4

GPU-centric implementation

This chapter provides much of the implied reasoning behind some of the choices made in Chapter 2, as they lend themselves particularly well to the present topic: GPU-centric disperse particle flow simulation. We have released our implementation of these methods, written in a combination of C and Cuda, as an open-source project available at `http://PhysalisCFD.org`. All calculations are presently performed using double precision floating point operations because the Poisson solver fails to converge using single precision. Owing to the significant GPU performance improvements that result from using single precision operations, it is highly likely that the implementation would benefit from adopting mixed-precision operations, though this has yet to be explored. We think of the current work as an intermediate step towards large-scale many-GPU computations that combines the traditional OpenMP and MPI parallelization methods, such as domain decomposition, at the higher levels

with GPU-centric methods at the lower levels of computation in order to fully utilize the resources available on many current high performance computing systems. The majority of the content of this chapter is adapted from Sierakowski (2016a).

4.1 GPU-centric parallelization

Parallelizing an algorithm for GPU operation presents unique challenges when compared to other methods of parallelization, such as OpenMP or MPI, because of a GPU's massively parallel architecture. As the names imply, the differences between a graphics processing unit (GPU) and a central processing unit (CPU) stem from the historical applications of each device. Generating graphics on a screen inherently requires the high-speed data-parallel calculation of many thousands to millions of independent pixels, which leads to the architecture we see today that prioritizes bandwidth over latency. Thus, making broad generalizations, CPU and GPU architectures differ primarily in the number and computational capability of logical cores and in the memory hierarchy. While a CPU will contain one to eighteen logical cores, each running between 2 and 3 GHz, a modern GPU may be comprised of more than 5,000 logical cores that each run just under 1 GHz. This is made possible by dedicating resources to computations at the expense of resources typically used to control work flow. In order to enable each of these many logical cores to access data, the memory layout of a GPU is significantly more complex than that of a CPU, as logical

cores can access a number of different memory spaces (global, shared, constant, texture, registers, e.g.) at vastly different speeds. As a result, writing code for a GPU often requires more development effort than code for a CPU, but given an appropriately data-parallel application, the resulting computations can run up to two orders of magnitude faster (Govindaraju et al., 2008; Silberstein et al., 2008).

When running a program on a GPU, the primary data resides in the dedicated onboard memory space—referred to as *global memory*— and it must be copied across the PCI Express system bus from the CPU memory. This system bus presents the largest hurdle for GPU computing as the transfer rate across it from the *host*—the CPU—to the *device*—the GPU—is limited to 0.5 GB/s (for PCI Express 2.0), which is a rate significantly smaller than the memory bandwidth of 288 GB/s (for an Nvidia Tesla K40) on the GPU.

Recognizing the severe limitation that device-host communication poses to the throughput of a GPU code, we take a *GPU-centric* approach to our program development: we seek to avoid all device-host data communication. We contrast this to a *GPU-accelerated* approach, which offloads the most data-parallel (or, often, the most convenient) pieces of a (legacy) CPU code onto a GPU. Under a GPU-accelerated approach, though a particular algorithm ported to the GPU may experience accelerations of up to 100 times, frequent device-host communication will often consume much of the time saved. By taking a GPU-centric approach, we avoid this device-host data communication completely and realize computational throughput much nearer

the actual speed of the GPU hardware.

To solve a generic spatial-temporal partial differential equation (PDE) (we will focus on a finite-difference solution to the Navier-Stokes equations below), our desired GPU-centric work flow is:

1. Define the initial condition, boundary conditions, coefficients, and forcing in host memory;

2. Copy all data to the device from the host;

3. Solve the PDE exclusively using the GPU without moving any data across the system bus;

3′. If desired, copy intermediate solutions to the host from the device for output to file;

4. Copy the final solution to the host from the device for output to file.

Under this GPU-centric approach, the CPU orchestrates the operation of the GPU by configuring and launching *kernels*—GPU code—but it does not operate on the PDE data. After the initial problem specification, the PDE solution proceeds exclusively on the GPU without any device-host data communication. We draw specific attention to Step 3′, which provides the ability to save intermediate solution time steps to file, though our GPU-centric work flow can proceed without any host-device data communication at all.

Compared to traditional programming methodologies, this GPU-centric approach represents a paradigm shift towards data-parallel operation. Recognizing that the design of the underlying computing hardware in many ways steers the choice of numerical methods applied, we choose our methods carefully and deliberately to take full advantage of that hardware. For example, a GPU contains multiple *streaming multiprocessors*—essentially, groups of computational cores—that access thousands of *registers* used to schedule thousands of *threads* that access *shared memory* at significantly higher rates than the *global memory*, which is accessible by all cores from all streaming multiprocessors. The Cuda GPU programming interface, which we use exclusively in this work, mimics this structure by grouping threads into *threadblocks* when configuring a kernel call; each threadblock corresponds roughly to a streaming multiprocessor. This design introduces two important concerns for programming a GPU: (i) using shared memory efficiently and (ii) avoiding *thread divergence*, which occurs when the kernel being run by a streaming processor splits into different paths due to a conditional statement. Since all of the computational cores inside a streaming multiprocessor must perform the same operations simultaneously, thread divergence is particularly detrimental because the streaming multiprocessor reverts to completing the kernel in a serial manner.

4.2 Flow solver parallelization

We describe here the implementation of the flow solver described in Section 2.1. A simulation runs in a GPU-centric manner on a single GPU; the 12-GB memory space available on an Nvidia Tesla K40 allows for up to $298^3 \approx 26.5 \times 10^6$ discrete cells (the CUSP sparse matrix library (Dalton et al., 2014) used to solve the pressure-Poisson problem requires a memory 100% overhead).

The methodology put forth in this section, especially the ideas behind *GPU parallelization* on a fixed grid discussed in Section 4.2.1, may be generalized to the solution of any PDE on a fixed grid. Section 4.2.2 describes the benefits of the use of ghost cells for assisting with computations near boundaries and Section 4.2.3 introduces a method for joining simulations in two separate GPUs using MPI communication.

4.2.1 Finite differences

Let us consider specifically one subset of data-parallel algorithms required for solving a generic PDE, such as the Navier-Stokes equations (1.1). The PDE may contain a spatial derivative such as the convective term of (1.1),

$$[(\mathbf{U} \cdot \nabla)\,\mathbf{U}] \cdot \mathbf{e}_x = (\nabla \mathbf{U}\mathbf{U}) \cdot \mathbf{e}_x = \frac{\partial\,(UU)}{\partial x} + \frac{\partial\,(UV)}{\partial y} + \frac{\partial\,(UW)}{\partial z}, \tag{4.1}$$

which we discretize as

$$\begin{aligned}[(\nabla \mathbf{U}\mathbf{U}) \cdot \mathbf{e}_x]_{i,j,k} \approx & \frac{(U_{i+1,j,k} + U_{i,j,k})^2 - (U_{i,j,k} + U_{i-1,j,k})^2}{4\Delta x} \\ & + \frac{(U_{i,j+1,k} + U_{i,j,k})(V_{i,j+1,k} + V_{i-1,j+1,k}) - (U_{i,j,k} + U_{i,j-1,k})(V_{i,j,k} + V_{i-1,j,k})}{4\Delta y} \\ & + \frac{(U_{i,j,k+1} + U_{i,j,k})(W_{i,j,k+1} + W_{i-1,j,k+1}) - (U_{i,j,k} + U_{i,j,k-1})(W_{i,j,k} + W_{i-1,j,k})}{4\Delta z}\end{aligned} \tag{4.2}$$

in Cartesian coordinates, operating on vector field variable (velocity) $\mathbf{U} = (U, V, W)$, where

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + \begin{pmatrix} i \\ j \\ k \end{pmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix}, \tag{4.3}$$

with (x_0, y_0, z_0) a reference corner of the domain, (i, j, k) integer-valued discretization indices, and $(\Delta x, \Delta y, \Delta z)$ discretization lengths in each direction. Note that (4.2) is formulated to account for interpolation required by our use of a staggered grid flow solver.

In order to compute the value of the derivative at each discrete point in space, $[(\nabla \mathbf{U}\mathbf{U}) \cdot \mathbf{e}_x]_{i,j,k}$, we must visit each point and use its value and those of the surrounding points from three planes of U ($i+1$, i, and $i-1$) and two planes of V (i and $i-1$) and W (i and $i-1$). The planes and their index references are depicted in Figure 4.1. This operation lends itself to one obvious GPU parallelization scheme: dedicate one GPU thread to each discrete point and compute the derivative by accessing the data of the neighboring points using each thread.

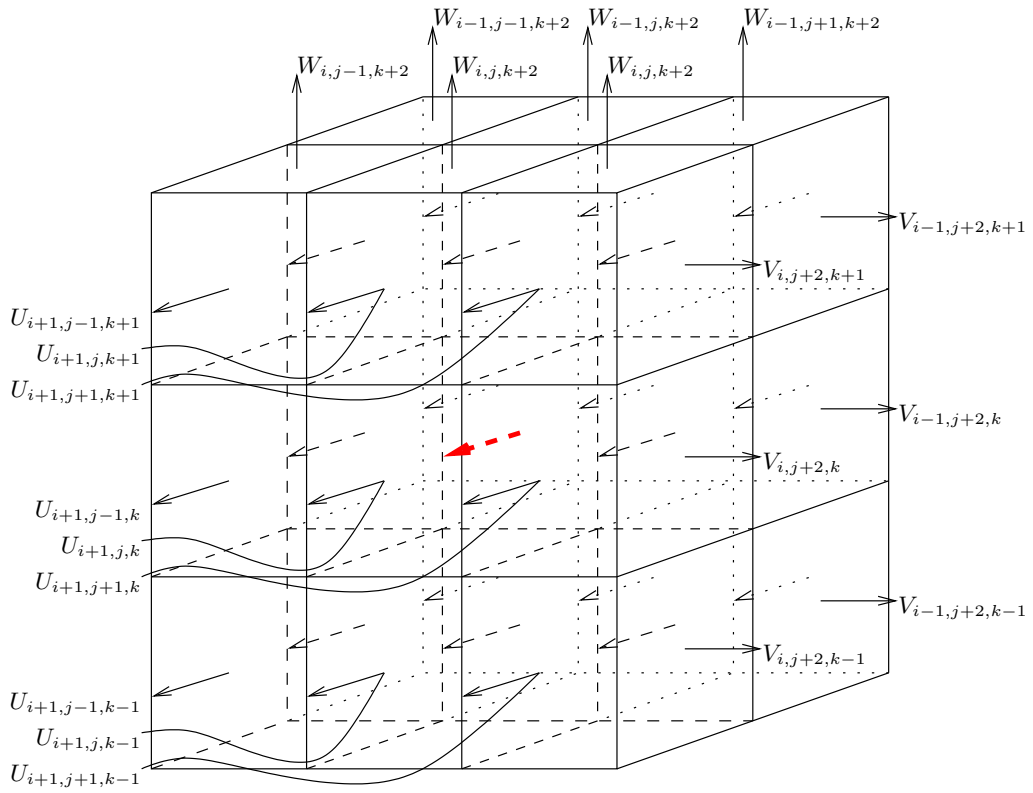


Figure 4.1: The staggered-grid velocity field discretization used in calculating finite differences on the GPU. The highlighted arrow in the center is $U_{i,j,k}$, the location at which the stencil is centered. Note that many internal vectors have been removed for clarity.

The GPU hardware permits the implementation of this general idea, but it requires a slightly more complex approach, especially since GPU shared memory proves valuable in algorithms that access the same memory many times. The maximum allowable sizes of threadblocks, as well as the availability of registers and shared memory, prevents us from implementing the simple kernel configuration of breaking the domain up into three-dimensional threadblocks. Instead, we use two-dimensional

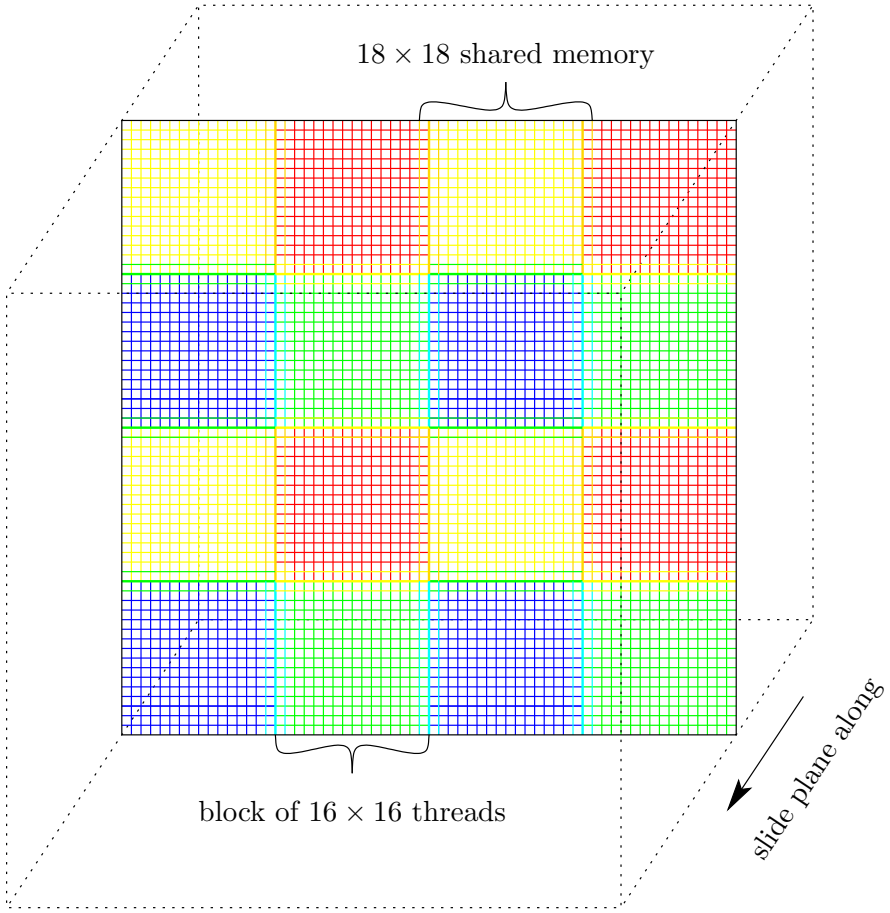


Figure 4.2: Laplace operator grid GPU parallelization scheme. Thread blocks are denoted by solid colors, with overlapping shared memory ghost cells denoted by combined colors. This data-parallel plane slides along the third Cartesian direction inside the GPU kernel.

threadblocks in the manner depicted in Figure 4.2:

1. Choose a predominant direction in which to perform the computation;

2. Divide the domain into two-dimensional blocks of threads (we use 16×16) in a plane normal to this predominant direction;

3. Launch the kernel that computes the derivative;

   (a) Slide along the predominant direction;

   (b) Copy the appropriate planes of U, V, and W to shared memory from global memory;

   (c) Compute (4.2);

We find that, by choosing a predominant direction and sliding the two-dimensional data-parallel plane along it, we efficiently use the available GPU hardware. The 16×16 threadblock size strikes a good balance between hardware utilization and (register and shared memory) resource availability. This scheme represents the first of two GPU parallelization paradigms essential to this work, namely, GPU parallelization on a fixed grid.

For the convective term, as well as other differential operators such as Laplacians and gradients, using shared memory expedites the computations since (4.2) indicates that each discrete data point will be read up to 15 times by 15 different threads. In Figure 4.3, we see that the ratio of the duration of the kernel run time for the global memory kernel to that of the shared memory kernel for this operation is greater than 2.5 for the largest resolution. This ratio increases initially as the problem grows to fill the GPU hardware until $N = 128$ but then stops increasing as the GPU has become fully saturated with work. Note that using shared memory is not always the best choice: a global memory read is required to copy data to shared memory, and

it becomes valuable only when reading data multiple times. We will apply this basic scheme to various operations in the flow solver parallelization below.

4.2.2 External boundary conditions

In order to avoid thread divergence, we take care to ensure that all threads perform the same operations at all grid nodes. When a finite difference stencil—e.g. the convective derivative (4.2)—must be computed at the domain boundary, such as the location $(i = 0, j, k)$, the code should behave differently because the data in the $(i - 1)$ position does not exist and therefore cannot be read. As such, we use ghost cells to simplify and standardize our differential operator calculations, which buffers

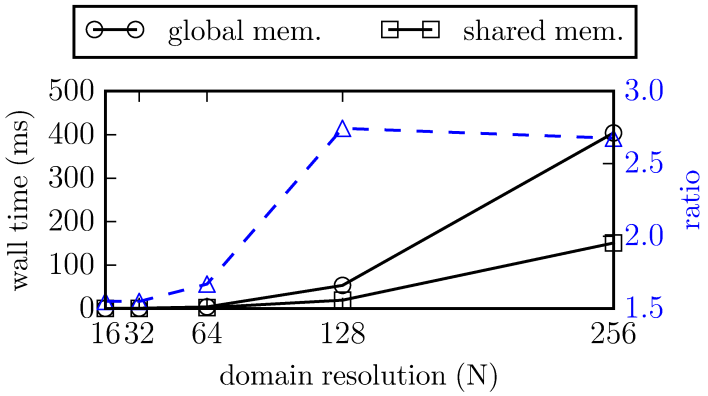


Figure 4.3: The runtime associated with using shared memory to compute the convective derivative (4.2) for various domain resolutions (the total number of discrete points is N^3). The ratio is the quotient of the wall time for the global memory implementation over the wall time for the shared memory implementation.

the field variable data structure with extra data points across the boundary so that the stencil may be computed on the boundary using the same set of operations as at every other point in the domain. Our method generally proceeds by performing the finite-difference operations required everywhere in space and then overwriting the results on the boundaries and in the ghost cells in order to adhere to the appropriate boundary conditions. These functions that enforce the boundary conditions follow the general parallelization scheme described above but work only on the boundary plane of data without sliding across the entire domain.

4.2.3 Concurrent precursor

In conjunction with a typical inflow/outflow simulation with Dirichlet velocity boundary conditions on the inflow—e.g. $\mathbf{U} = \left(\bar{U}, 0, 0\right)$—Neumann conditions on the outflow, and periodic conditions on the four remaining boundaries, we have implemented a mechanism for introducing turbulent perturbations $\mathbf{U}'$ to the inflow. We generate the turbulent perturbations by concurrently simulating triply-periodic homogeneous isotropic turbulence on a separate GPU (or, if desired, on the same GPU at the expense of simulation size) and sampling the velocity field from it. In this domain, the turbulence is maintained using the linear forcing of Rosales and Meneveau (2005) with an additional enhancement provided by Carroll and Blanquart (2013). We refer to this homogeneous isotropic turbulence simulation as the *precursor domain* and the inflow/outflow simulation as the *experimental domain*. Using a plane

parallel to the inflow boundary of the experimental domain called the *perturbation plane*, we slide across the precursor domain at the desired inflow velocity in such a way to introduce the mean flow—e.g. $\mathbf{U} = \left(-\bar{U}, 0, 0\right)$—and interpolate the precursor domain velocity field to the perturbation plane (see Botto and Prosperetti, 2012). We use the turbulent perturbations, which are now added to the mean inflow velocity, as the inflow Dirichlet boundary conditions—e.g. $\mathbf{U} = \left(\bar{U} + U', V', W'\right)$. Figure 4.4 illustrates this mechanism. The two simulations proceed together in time using identical time step sizes. The maximum Taylor-scale Reynolds number $\mathrm{Re}_\lambda = \lambda \left|\mathbf{U}'\right| / \nu$ that we have successfully simulated in this way is $\mathrm{Re}_\lambda \approx 100$ on a grid of size 298^3.

In order to enforce the precursor turbulent perturbations on the experimental inflow velocity, we communicate between the two GPUs using the following procedure:

1. Interpolate the precursor velocity field to the perturbation plane;
2. Copy the perturbation plane to the precursor host from the precursor device;
3. Using MPI, communicate the contents of the perturbation plane to the experimental host;
4. Copy the perturbation plane to the experimental device from the experimental host;
5. Apply the perturbation plane to the inflow boundary condition.

In order to optimize the code by controlling the timing of data transfers and synchronizing them with computations, we perform these steps explicitly instead of relying

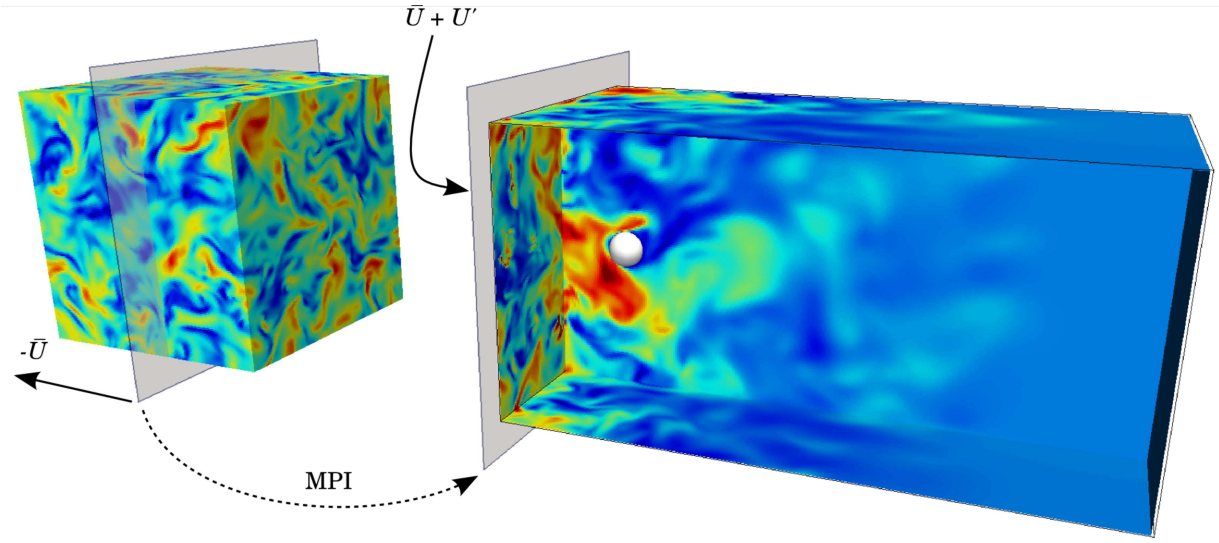


Figure 4.4: The concurrent precursor. The plane cutting through the homogeneous isotropic turbulence precursor simulation translates through the domain at the velocity of the mean inflow for the experimental domain. The turbulent velocity field from the precursor plane provides turbulent fluctuations about the mean flow for the inlet to the experimental domain via MPI communication between two GPUs.

on Cuda unified memory, which limits precise control.

One concern about the use of this method is that it forces the fluid directly at all scales, which may obscure the effect of the particles. In order to avoid this potential difficulty, Physalis particles may only be introduced into the experimental domain, where simulations often take the form of decaying turbulence, as if in a wind tunnel with decaying grid-induced turbulence.

4.3 Physalis method parallelization

In addition to storing the particle data such as material properties and Lamb's coefficients in a Lagrangian sense, we maintain five Eulerian field variables spanning the entire domain that assist with applying particle boundary conditions while avoiding thread divergence (see Section 4.3.2). The first, `phase`, is a cell-centered phase mask where the value is set to either the particle number if the center of the cell is contained within a particle and -1 otherwise. This field is not meant to reconstruct the particle volume perfectly, but instead to denote which fluid cells are contained within the particle cage. A related field, `phase_shell`, is cell-centered and has a value of 0 everywhere except for the outermost set of cells in `phase`, where it is set to 1. This field specifies the location of the pressure component of the particle cage. The remaining three fields constitute a vector field denoting the locations of the velocity components of the cage. Each of these face-centered components, `flag_{u,v,w}`, is set equal to 1 everywhere except for the locations of the velocity components of the cage, where it is set equal to -1 (it is also used for specifying external domain boundary conditions, where it is set to 0).

As currently implemented, the number of particles n_p in a simulation remains constant for the duration of the simulation. Particles may cross periodic boundaries, but they cannot enter or leave the domain through Dirichlet or Neumann boundaries. Further, the particle translation and rotation flags enable the user to define whether a particle may move freely about the domain, move without rotation, rotate without

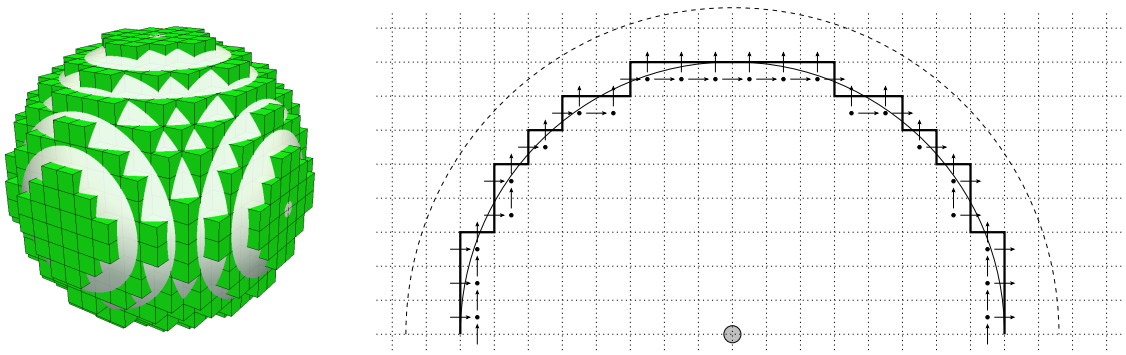
moving, or remain fixed in space.

This section focuses specifically on the Physalis method and also provides in Sections 4.3.1 and 4.3.2 examples of the general grid parallelization paradigm discussed above. Section 4.3.3 discusses our GPU parallelization of the Lebedev quadrature integration scheme and Section 4.3.4 discusses the more nuanced details regarding the decoupling of the fluid outside the particle from that inside the particle. Finally, Section 4.3.5 introduces the second GPU parallelization paradigm, *particle parallelization*, which may be generalized to any particle-based numerical method.

4.3.1 Particle cage construction

At the beginning of every time step, we reconstruct the particle cages described in Section 2.2—that is, we fill in the `phase`, `phase_shell`, and `flag_{u,v,w}` fields—in order to update their shapes according to the new positions of the particles. Figure 4.5b illustrates the locations denoted by `phase_shell` and `flag_{u,v,w}` as dots and arrows, respectively. As governed by the time step size, particles move, at most, a fraction of a discrete cell at any time step, so that it is not strictly necessary to reconstruct the cages at every time step. However, we choose to perform the reconstruction at every time step in order to limit the magnitude of the force fluctuations that appear as a result of changes in the cage shape (see this effect in Section 5.3). While it may seem appropriate to reconstruct only the cages of particles that have moved and to reuse the previous cages of the remaining particles, we draw particular

attention to the deficiency underlying this idea in a more direct setting in Section 4.3.2.



(a) Three dimensional representation. The cage is a closed surface, part of which cannot be seen as it resides inside the particle.

(b) A two-dimensional plane through the center of the Physalis cage in (a). Velocity and pressure boundary conditions are denoted by arrows and dots, respectively. The dashed line represents the integration surface used for the scalar products (2.21).

Figure 4.5: An example of the Physalis particle cage that specifies the locations of the analytic boundary conditions.

We generate a particle cage using the following routine:

1. Mark all cells of `phase` with cell centers inside the particle surface with the particle number (and -1 if fluid); the outermost shell of these cells, `phase_shell`, constitutes the set of cage cells;

2. The centers of the `phase_shell` cage cells, marked with 0 if fluid and 1 otherwise, are the locations at which the Dirichlet conditions for the pressure-Poisson

equation are imposed;

3. All faces of the cage cells `flag_{u,v,w}` with an outward normal with respect to the particle are part of the set of points where the velocity boundary conditions are imposed, and are marked with -1 (and 1 otherwise);

4. The set of velocity nodes is completed by including the centers of all faces common to two neighboring cage cells by marking them with -1.

Thus, reconstructing the cages requires one sweep of the domain for each cage field variable, and then one extra for each of the `flag_{u,v,w}` in order to complete Step 4. This reconstruction technique, being fully GPU-parallelized using the grid parallelization scheme described in Section 4.2.1, typically constitutes less than 2% of the total runtime and significantly reduces force fluctuations when applied at every time step.

4.3.2 Internal boundary condition application

Though unintuitive at first glance, the prioritization of bandwidth over latency in GPU hardware means that we can sometimes achieve faster performance overall by doing a significant amount of extra work. Generally speaking, when performing a serial (or lightly parallelized) calculation, limiting the amount of work to be done is almost always the correct choice. When performing a GPU (or massively parallelized) calculation, however, this is not always the correct answer because it may lead to

underutilization of the hardware or strong thread divergence thereby increasing overall runtime.

When applying the particle boundary conditions (2.15) at the locations designated by the cage with `phase_shell` and `flag_{u,v,w}`, we find that it is an inefficient use of the GPU hardware to limit the work to be done by updating only the subset of the flow velocity and pressure nodes denoted by the cages. Our boundary condition application algorithm applies the grid parallelization scheme of Section 4.2.1 instead of the more obvious, but less efficient, particle parallelization scheme described below. The particle boundary condition application works in this way:

1. For each pressure and velocity field component, visit each grid cell in the manner of the grid parallelization scheme;

2. For each thread, upon arriving at a given cell, determine to which particle the cell belongs (if any) using `phase`;

3. In a temporary location, calculate the boundary condition to be applied using (2.15) depending on the particle number and, if the cell is a fluid cell, perform the same calculations using a dummy particle;

4. Write the temporary value to the cell only if the cell is a cage node, according to `phase_shell` and `flag_{u,v,w}`.

By applying the grid parallelization scheme and visiting all velocity cells, we achieve a faster overall boundary condition application algorithm despite doing a significant

amount of work that is ultimately discarded because we fully utilize the GPU hardware and avoid thread divergence.

In order to avoid thread divergence in Step 4, we apply the following useful technique that effectively performs an `if/else` operation without branching by using common operations on all threads. Here, as a simple example, we determine whether to write the analytic pressure boundary condition at location `C`. If there is a particle cage node at this location—i.e., `phase_shell` is 1—the first term of the formula will be nonzero and pressure will be overwritten with the analytic solution (2.14b). Otherwise, `phase_shell` is 0 so the second term will be nonzero and the pressure will remain unchanged:

```
p[C] = phase_shell[C] * p_analytic + (1 - phase_shell[C]) * p[C].
```
(4.4)

4.3.3 Lebedev quadrature scalar products

Lamb's coefficients are key to coupling between the particle and flow solutions. They determine the velocity and pressure boundary conditions (2.15) applied at the cage, which the flow solution must obey. In turn, Lamb's coefficients contain information derived from the flow solution, and we determine their values through scalar products (surface integrals) of the form (2.21).

As explained in 2.5, the coefficients p_{lm}, ϕ_{lm}, and χ_{lm} are calculated by taking

scalar products of the type

$$(Y_l^m, \tilde{p}) = \int_{-\pi}^{\pi} d\varphi \int_0^{\pi} \sin\theta \, d\theta \, \bar{Y}_l^m(\theta, \varphi) \, \tilde{p}(\theta, \varphi), \tag{4.5}$$

where the overbar denotes complex conjugation. We have found that the use of an integration scheme based on Lebedev quadrature (Lebedev, 1976) in place of the differential angular integration scheme used in G&P provides a more efficient algorithm for the evaluation of these integrals.

A Lebedev quadrature rule of order l integrates exactly spherical harmonics up to order l at the expense of function evaluations at a relatively small number of nodes on the unit sphere. In most of our simulations we use a maximum order of spherical harmonics $l = 3$ (and occasionally $l = 4$) and therefore, ideally, we would need an integration rule capable of integrating spherical harmonics of order 6 (or possibly 8). Lebedev rules are available for orders 5, 7, 11, and higher. We have chosen the seventh-order rule that requires only 26 quadrature nodes as opposed to 50 nodes for the 11th-order rule; it takes the form

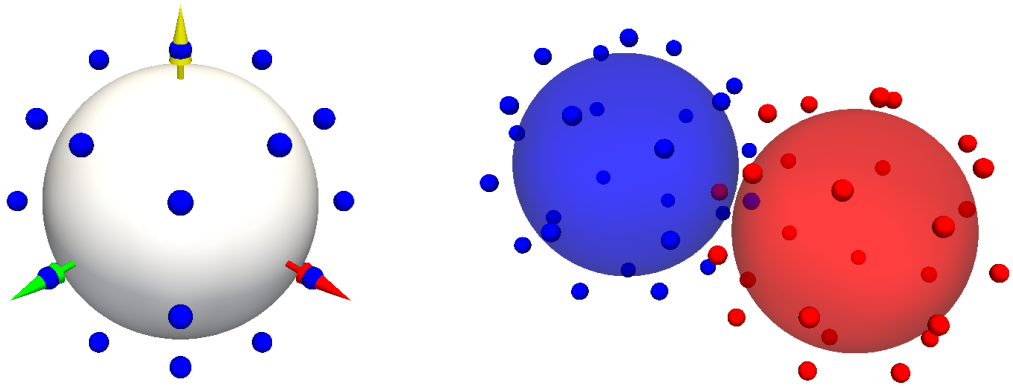
$$\frac{1}{4\pi} \int_{-\pi}^{\pi} d\varphi \int_0^{\pi} \sin\theta \, d\theta \, f(\theta, \varphi) \approx A_1 \sum_{i=1}^{6} f\left(a_i^1\right) + A_2 \sum_{i=1}^{12} f\left(a_i^2\right) + A_3 \sum_{i=1}^{8} f\left(a_i^3\right). \tag{4.6}$$

The Lebedev quadrature node weights $\{A_1, A_2, A_3\}$ and positions $\{a_i^1, a_i^2, a_i^3\}$ are reproduced in Table 4.1 and illustrated in Figure 4.6a. We interpolate the flow field values at the quadrature nodes from the Cartesian grid using trilinear interpolation as described in G&P.

Since Lamb's coefficients depend on the flow field in the neighborhood of the

N	a_i^1	(θ, φ)	N	a_i^2	(θ, φ)	N	a_i^3	(θ, φ)
1	a_1^1	$\left(\frac{\pi}{2}, \frac{\pi}{2}\right)$	7	a_1^2	$\left(\frac{\pi}{2}, \frac{\pi}{4}\right)$	19	a_1^3	$\left(\alpha_1, \frac{\pi}{4}\right)$
2	a_2^1	$\left(\frac{\pi}{2}, \pi\right)$	8	a_2^2	$\left(\frac{\pi}{2}, \frac{3\pi}{4}\right)$	20	a_2^3	$\left(\alpha_1, \frac{3\pi}{4}\right)$
3	a_3^1	$\left(\frac{\pi}{2}, \frac{3\pi}{2}\right)$	9	a_3^2	$\left(\frac{\pi}{2}, \frac{5\pi}{4}\right)$	21	a_3^3	$\left(\alpha_1, \frac{5\pi}{4}\right)$
4	a_4^1	$(0, 0)$	10	a_4^2	$\left(\frac{\pi}{2}, \frac{7\pi}{4}\right)$	22	a_4^3	$\left(\alpha_1, \frac{7\pi}{4}\right)$
5	a_5^1	$(\pi, 0)$	11	a_5^2	$\left(\frac{\pi}{4}, 0\right)$	23	a_5^3	$\left(\alpha_2, \frac{\pi}{4}\right)$
6	a_6^1	$\left(\frac{\pi}{2}, 0\right)$	12	a_6^2	$\left(\frac{\pi}{4}, \frac{\pi}{2}\right)$	24	a_6^3	$\left(\alpha_2, \frac{3\pi}{4}\right)$
			13	a_7^2	$\left(\frac{\pi}{4}, \pi\right)$	25	a_7^3	$\left(\alpha_2, \frac{5\pi}{4}\right)$
			14	a_8^2	$\left(\frac{\pi}{4}, \frac{3\pi}{2}\right)$	26	a_8^3	$\left(\alpha_2, \frac{7\pi}{4}\right)$
			15	a_9^2	$\left(\frac{3\pi}{4}, 0\right)$			
			16	a_{10}^2	$\left(\frac{3\pi}{4}, \frac{\pi}{2}\right)$			
			17	a_{11}^2	$\left(\frac{3\pi}{4}, \pi\right)$	$\alpha_1 = \arccos\left(3^{-1/2}\right)$		
			18	a_{12}^2	$\left(\frac{3\pi}{4}, \frac{3\pi}{2}\right)$	$\alpha_2 = \arccos\left(-3^{-1/2}\right)$		
$A_1 = 0.598398600683775$			$A_2 = 0.478718880547015$			$A_3 = 0.403919055461543$		

Table 4.1: Lebedev quadrature nodes and weights to be used in (4.6). We choose θ to be the polar angle.



(a) The locations of the quadrature nodes used in the seventh-order Lebedev quadrature rule. The arrows represent the three Cartesian directions.

(b) An example of interfering Lebedev quadrature nodes, with the red node belonging to the red particle residing inside the blue particle (and likewise for the blue node).

Figure 4.6: Illustrations of Physalis particles with their Lebedev quadrature scalar product nodes.

sphere, the surface of integration for the scalar products must be located at a radial location r outside the sphere, but still within the region of validity for the Stokes approximation. We refer to the work of G&P for an analysis of the accuracy of various choices of the ratio of r to the particle radius a; for the range of Reynolds numbers that we have investigated, we find $1.1 < r/a < 1.25$ balances accuracy and numerical efficiency. We must take special care to treat the situation of particles sufficiently near to each other so that their Lebedev quadrature nodes interfere as illustrated in Figure 4.6b; we discuss the treatment of these cases in the following

section.

First, we will present our method of parallelizing Lebedev quadrature on a GPU. The quadrature nodes, attached to the particle, are free to reside at any location in space, while the flow data that they are integrating appear at discrete locations according to the flow solver grid. The first step, therefore, is to interpolate the flow data to the quadrature nodes. Given 26 nodes per n_p particles, we configure our first kernel call with n_p threadblocks of 26 threads each. For each thread, representing one quadrature node of one particle, we interpolate from the discrete field to the quadrature node using tri-linear interpolation and store the values in a temporary array.

In a subsequent step, we calculate each of the six scalar products (2.21), representing the three sets of complex-valued Lamb's coefficients. Given a truncation order $2 \leq L \leq 4$ for the reconstruction of Lamb's solution from the coefficients, each set requires $M = \frac{1}{2}(L+1)(L+2)$ coefficients, so we have $6M$ total coefficients to calculate for each particle (though some are identically zero). We change our kernel configuration to have a two-dimensional threadblock of size $(n_p \times 6M)$ of 26 threads each. The kernel call then proceeds as follows:

1. Calculate at a node the integrand of (2.21)

2. Ensure that all threads have completed their work;

3. Use the zeroth thread (associated with node zero) in each block to perform the

sum (4.6);

4. Use the zeroth thread of the second dimension of the threadblock (the $6M$ coefficients) to manipulate the surface integrals to yield Lamb's coefficients as in (2.24).

As the number of items to sum is small in all cases, we perform the sums directly in the kernel using a subset of the available threads because it forgoes the overhead associated with a parallel reduction.

4.3.4 Particle interior

In general, the flow solver operates on all fluid grid nodes in the domain, including those inside the particles. However, we make sure that the internal in no way affects the external flow by decoupling the pressure as explained below and by squelching any fluid motion by enforcing that $\mathbf{U}^*_\kappa = 0$ at these nodes in 2.25. As mentioned in Section 2.6, we subsequently replace the internal flow with the solid body motion of the particle. In past work of Z&P, internal flow was generated by the solution procedure because the process of solving the momentum equation (1.1) everywhere in space simplified the numerical algorithm. We have found in the current work, however, that the internal velocities generated in this way can be large, thus unnecessarily limiting the time step. The present method overcomes this limitation, but has some implications for our flow solver, which we discuss presently.

The fractional step method that we adopt for solving the Navier-Stokes equations (see Section 2.1) requires the solution of a pressure-Poisson equation of the form

$$\nabla^2 \phi_\kappa^{n+1} = \frac{\rho_f}{\Delta t} \nabla \cdot \mathbf{U}_\kappa^*. \tag{2.28}$$

where ϕ represents the pressure correction required to ensure that the solution to the momentum equation (1.1) satisfies the continuity equation (1.1b), and $\mathbf{U}^*$ is the intermediate velocity field (2.1). In Z&P, this equation was solved by imposing homogeneous Neumann conditions on the cage nodes as is usually done with the fractional step method. In the present work we have improved on this approximate procedure by imposing the analytically accurate pressure boundary condition (2.14) provided by Lamb's solution.

In addition to providing a more physically accurate boundary condition than the zero-normal-gradient condition, this new procedure carries with it a number of desirable numerical consequences as well:

1. Lamb's solution converges in fewer iterations for the same overall accuracy;

2. Posing Dirichlet boundary conditions on (2.4) removes the solvability condition constraint that appears in the case of Poisson problems with homogeneous Neumann boundary conditions;

3. As more particles are added to a simulation, the Poisson solver converges more rapidly since the solution gets specified at more points that are distributed throughout the domain. As a consequence, the condition number of the discrete

Laplace operator is decreased because the low frequency modes are suppressed. Strongly inhomogeneous particle distributions may affect this property;

4. If a particle is positioned so that there exists a single grid cell jutting out from the rest of the cage, the zero-normal-gradient is applied to five out of six cell faces, which causes difficulties in the solution of the Poisson problem so much so that it may fail to converge; switching to Dirichlet boundary conditions removes this difficulty.

In general, utilizing the analytical representation (2.14) provided by Lamb's solution improves the efficiency of Physalis by ensuring better consistency with the Navier-Stokes equations.

We apply the analytical Dirichlet boundary condition (2.14b) to the standard second-order finite difference matrix representation of (2.28) by setting the right-hand side of (2.28) equal to

$$\phi_\kappa^{n+1} = p_\kappa^{n+1/2} - p^{n-1/2} \tag{4.7}$$

at the designated cage nodes, where the $p_\kappa^{n+1/2}$ is equal to (2.14b) along with the change of reference frame. We then edit the corresponding rows and columns in the Laplace operator matrix to be equal to one on the diagonal at the designated cage nodes and zero elsewhere. The resulting matrix structure is shown in Figure 4.7 for a $4 \times 4 \times 4$ triply-periodic domain with one (severely under-resolved for illustration purposes) particle in the center. Here, the matrix elements corresponding to the

nodes marked in yellow have a value of one while the red nodes on the same row are set to zero. This method is extremely beneficial to the flow solver because it specifies physically accurate pressure boundary conditions at the particle surface.

Our solution procedure is now inherently independent of the particle interiors as long as particles are sufficiently separated. However, to maintain this independence when a particle approaches another one so that any of its Lebedev quadrature nodes fall inside the other particle as mentioned above, we should have recourse to a different integration scheme, which is undesirable from the perspective of code efficiency. To be able to continue using the Lebedev scheme, we set the interior velocity field inside every particle equal to the solid body (linear and angular) motion of the particle and the pressure (essentially) equal to the mean pressure within the Stokes region near the particle (namely, the p_{00} term in (2.14b)).

This scheme provides the most natural and well-behaved treatment of the particle interference situation because, due to the no slip condition, it guarantees a smooth transition of velocity as a quadrature node traverses the surface of a neighboring particle. This procedure requires no special treatment of these interfering quadrature nodes as they simply interpolate an appropriate value under all particle configurations. The effect of any residual error is mitigated by the fact that the fraction of offending nodes is typically very small (at most 1/26).

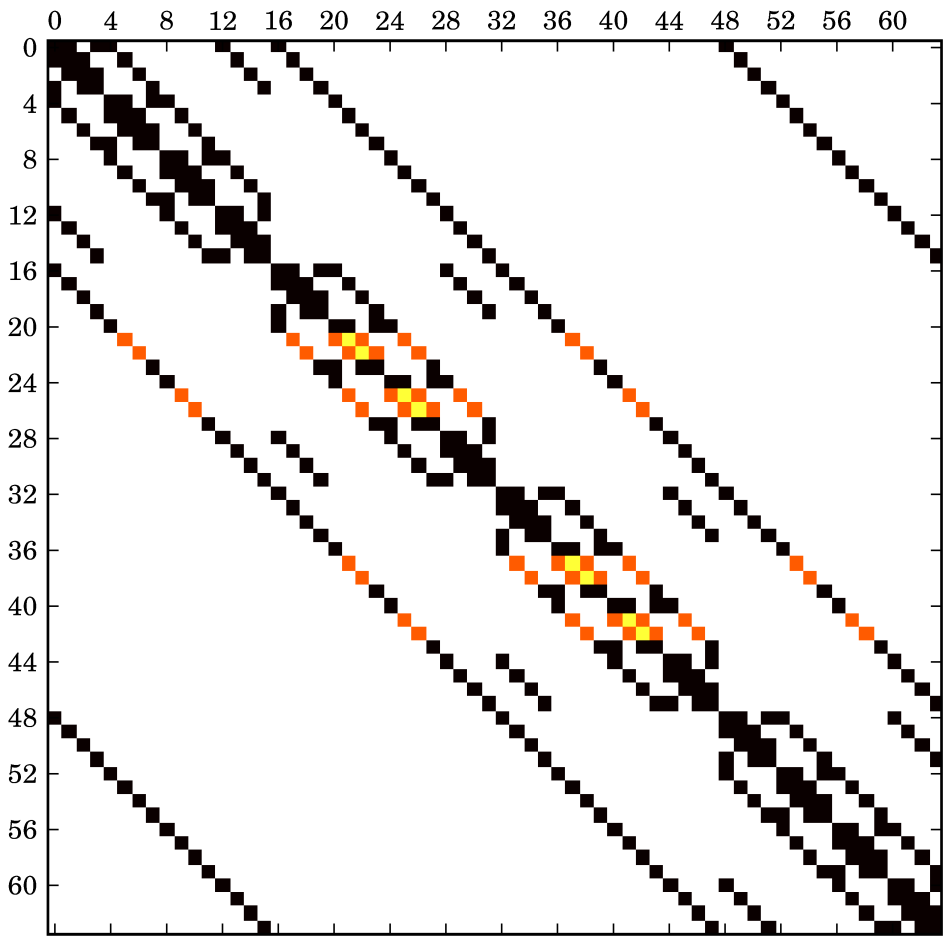


Figure 4.7: The decoupled pressure matrix for an example $4 \times 4 \times 4$ triply-periodic domain with one (severely under-resolved) particle in the center. Nodes that are not white are initially nonzero according to the typical second-order finite difference stencil. When applying the particle boundary conditions, black nodes are unchanged, and yellow and red nodes are set equal to one and zero, respectively.

4.3.5 Force calculation and trajectory integration

Once we have obtained Lamb's coefficients, the hydrodynamic force $\mathbf{F}_h$ and couple $\mathbf{L}_h$ are computed directly as explained in Section 2.7. We parallelize this computation by assigning a thread to each particle and computing $\mathbf{F}_h$ and $\mathbf{L}_h$ independent of the others. This represents an application of our second essential GPU parallelization paradigm, namely, GPU parallelization across particles.

The calculation of the interaction forces such as those discussed in Chapter 3 requires more attention, however. Both the lubrication and contact models require the calculation of the $O\left(n_p^2\right)$ particle-particle interaction n_p-body problem as all pairs must interact. Relying on the compact support of (3.11), we can ignore the modeled interaction between α and other particles that are far away (the fluid still carries long-range hydrodynamic interactions). In this way, we operate only on the neighbors of α within $\varepsilon \sim a_\alpha$ and significantly decrease the total number of required interactions.

We implement a GPU-centric n_p-body algorithm that relies on a spatial partitioning of the domain to generate lists of particles near enough to interact with each other (this is similar to a sample code distributed with the Nvidia Cuda Toolkit); we recognize this as an advanced application of our second GPU parallelization paradigm. It is important to draw attention to the wealth of research on the topic of n_p-body simulation of which the method described below is but an example. For more advanced and/or specialized methods, see, e.g., Fortin et al. (2011).

Consider the situation depicted in Figure 4.8 with 64 particles with finite radius

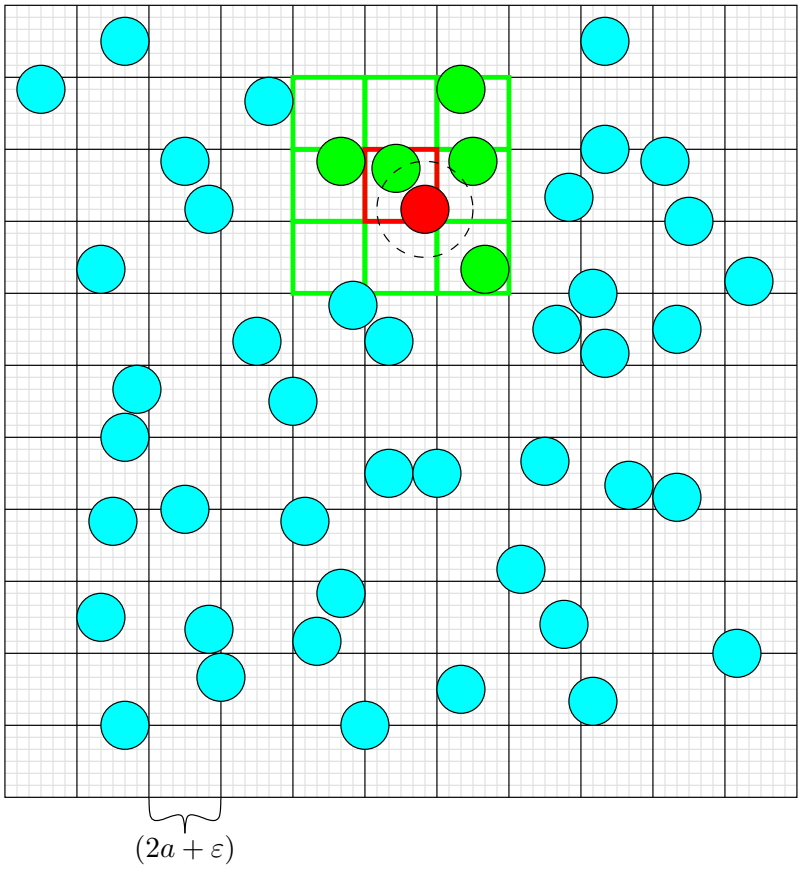


Figure 4.8: A two-dimensional abstraction of the nearest-neighbors interaction algorithm. The red particle must interact with all particles within ε (dashed radius), and the algorithm selects those in green. The red box contains the red particle's center, and it must be interacted with all particles whose centers fall inside the eight neighboring boxes (26 in three dimensions), highlighted in green.

a distributed throughout a domain, uniquely numbered zero through 63. Taking ε to be the compact support length beyond which (3.11) may be safely ignored, we divide the domain into segments called *bins* of size $2a + \varepsilon$, which represents the minimum distance required to ensure particles residing in two adjacent bins properly interact.

We set up an empty list of length n_p that will associate bin numbers with the particles they contain. We then proceed as follows:

1. Thread over all particles to determine in which bin the center of each particle resides;

2. Carrying along the associated particle number, sort the particles by bin index using a GPU sorting algorithm;

3. Thread over all bins to determine the first and last particle in each bin (to assist with parallelization);

4. Thread over all particles to compute (3.11);

   (a) Loop over all particles residing in the same bin as the particle in question;

   (b) Loop over all particles residing in the 26 neighboring bins;

   (c) Sum the influence of all neighboring particles.

Following this process, we observe $O(n \log n)$ complexity, significantly improving upon the $O(n^2)$ all-pairs algorithm. Figure 4.9 illustrates the superiority of the spatial partitioning algorithm by varying the number of randomly positioned particles in a domain of constant size. The dashed lines show that the spatial partitioning method significantly improves upon the run time of the all-pairs method, with gains recognized at as few as $n_p = 8$ and a factor of nearly 10,000 improvement for $n_p = 4096$. The solid lines indicate that for simulations with more than approximately $n_p = 500$,

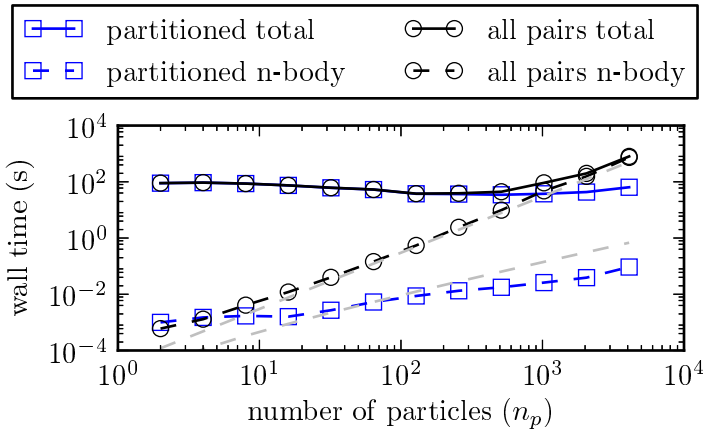


Figure 4.9: A comparison of the spatial partitioning and all-pairs algorithms by varying the number of randomly placed particles in a domain of constant size. The computation times for both the n_p-body function alone and the total simulation are shown. Overlaying the plots are dashed lines representing $O(n^2)$ and $O(n \log n)$ scaling.

the brute-force all-pairs calculation is more time consuming than the solution of the pressure-Poisson problem, and that the spatial partitioning method yields more than a factor of 10 improvement for the overall simulation with $n_p = 4096$. Further, we see that this method appears to perform slightly better than the expected $O(n \log n)$ complexity because the GPU spends, proportionally, less time launching kernels and more time computing as more particles are added. Using the space-partitioned particle-based operation scheme, in contrast to the cage generation and boundary condition application examples above, we find a situation where limiting the total amount of work does indeed pay off and significantly improves the overall performance, but only

because we are able to saturate the GPU hardware with computations.

4.4 Computational efficiency

In order to discuss the computational efficiency of this GPU-centric implementation of the Physalis method, we compare it to the best available alternative: a legacy implementation of the method. This legacy implementation, which follows the approach of Zhang and Prosperetti (2005) with the scalar product improvements of Gudmundsson and Prosperetti (2013), is a combination Fortran and C++ code optimized for serial CPU operation. We recognize the myriad limitations underlying comparisons of GPU-optimized codes with serial-CPU-optimized codes, but we remain convinced of the value of this study provided we sufficiently control for those limitations.

It is important to note that there are more differences between the GPU and the CPU codes than simply the hardware on which they run, as each code is optimized for its respective hardware. In addition to low-level differences such as choosing underlying data structures best suited for each programming model, there also exist significant differences in the high-level numerical methods employed in each code. As such, we perform a series of simulations controlled to make the comparisons as fair as possible.

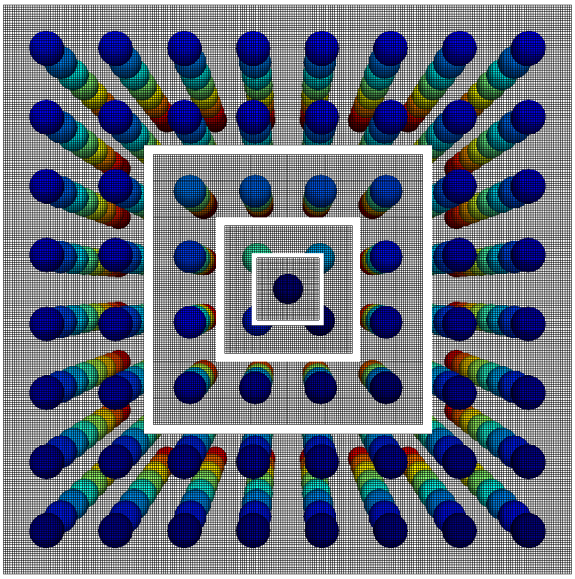
We perform ten simulations of various size. Each simulation uses the following

conditions:

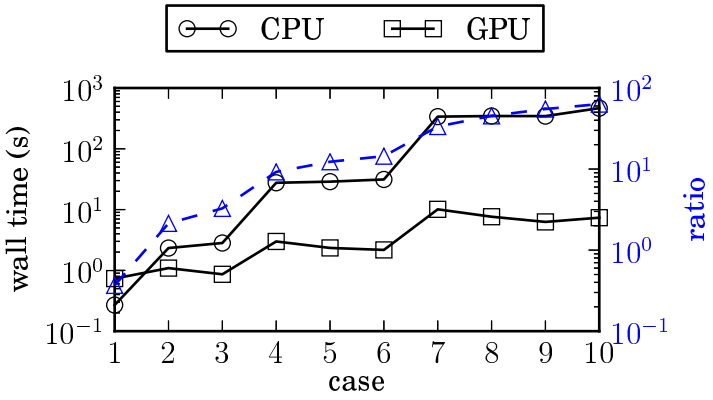
- Periodic boundary conditions in all three directions of a box of size L^3;
- Initially quiescent flow driven by a normalized pressure gradient $a^3\mu^{-1}\nu^{-1}\nabla p \cdot \mathbf{e}_z = 10$;
- Fixed particles (no translation or rotation) distributed on a uniform grid with resolution $a/\Delta x = 8$.

We vary the flow domain resolution $(L/\Delta x)^3$ from 32^3 to 256^3 by powers of two, with arrays of one, eight, 64, and 512 particles for each, if they fit into the domain with at least a particle diameter separation. For example, a 32^3 domain can only fit one particle at resolution $a/\Delta x = 8$, so we perform only one such simulation. In the 256^3 domain, however, we can fit up to 512 particles, so we perform simulations with all four sets of particles. For all simulations, the particles are distributed in such a way that they evenly fill the entire domain; the 256^3 simulation with 512 particles looks exactly the same as if we repeated the 32^3 simulation with one particle and tiled it eight times in each of the Cartesian directions. Four of these configurations are illustrated in Figure 4.10a.

We run the simulations for four time steps using each code with a time step size of $\Delta t \approx 0.0013$, which corresponds to a CFL number of approximately 0.5 (Ferziger and Perić, 2002). We limited the number of time steps to four because of the time required to run the largest cases on the CPU. We consider the average timing results



(a) A representation of the ten simulations used for comparison. Each simulation varies in resolution along each side and in number of particles.



(b) The iteration time $\bar{t}_i$ and CPU-to-GPU ratio from Table 4.2.

Figure 4.10: The comparison between the current work on a single GPU and a legacy serial CPU implementation of the Physalis method.

in Table 4.2, where we denote each simulation with a number and a letter—C for CPU code and G for GPU code. Given the differences between the numerical methods underlying each code, we record the number of iterations required for Lamb's coefficient convergence N_i and report averages for the runtime of each code as both iteration $\bar{t}_i$ and time step $\bar{t}_s$ averages. For the GPU code, we list the average time for initial condition and intermediate solution data transfers $\bar{t}_D$ in addition to the total run time t_T. The data transfers happen once in each direction during t_T, which, with only four time steps, over-emphasizes their effect as we typically require the storage of intermediate time steps as infrequently as every 10 to 100 time steps, depending on the simulation. The final column of Table 4.2 lists the ratio of the run time for the CPU to the run time for the GPU to provide an estimate of their relative computational efficiency.

Every simulation is run on an in-house development cluster with the following relevant specifications:

- Intel Xeon E5-2650v2 CPU;
- DDR3 1866 MHz RAM;
- Nvidia Tesla K40 GPU.

Both codes run with the same numerical accuracy convergence parameters, and as shown in Chapter 5, the GPU code is at least as accurate as the CPU code, and is even superior in some measures.

case	$\left(\frac{L}{\Delta x}\right)^{\frac{1}{3}}$	n_p	timing (s)		averages (s)			ratios (C/G)	
			$\bar{t}_D$	t_T	N_i	$\bar{t}_i$	$\bar{t}_s$	t_T	$\bar{t}_i$
1C	32	1	–	6.18	23	0.269	1.55	0.351	0.367
1G			0.0922	17.6	24	0.733	4.40		
2C	64	1	–	53.7	23	2.33	13.4	2.59	2.14
2G			0.151	20.7	19	1.09	5.19		
3C	64	8	–	67.6	24	2.82	16.9	3.57	3.27
3G			0.165	19.0	22	0.862	4.74		
4C	128	1	–	636	23	27.6	159	12.5	9.24
4G			0.715	50.9	17	2.99	12.7		
5C	128	8	–	690	24	28.8	173	15.5	12.3
5G			0.722	44.4	18	2.34	11.1		
6C	128	64	–	1470	47	31.3	368	37.8	14.5
6G			0.732	39.0	18	2.17	9.75		
7C	256	1	–	7760	23	338	1940	45.1	33.3
7G			2.91	172	17	10.1	43.1		
8C	256	8	–	8290	24	346	2070	63.9	45.3
8G			3.13	130	17	7.63	32.4		
9C	256	64	–	14100	41	345	3530	125	54.9
9G			3.38	113	18	6.27	28.2		
10C	256	512	–	21900	47	466	5480	157	63.3
10G			3.58	133	19	7.36	34.9		

Table 4.2: A summary of the results of the computational efficiency comparison between the current GPU-centric implementation and the legacy serial CPU implementation of the Physalis method. The case number denotes whether the simulation was performed using the CPU (C) or the GPU (G). Here, $\frac{L}{\Delta x}$ is the resolution on each side of the domain, n_p the number of particles, $\bar{t}_D$ the average data transfer time, t_T the total runtime, N_i the number of Lamb's iterations for all four time steps, and $\bar{t}_i$ the time for one Lamb's iteration.

We present in Table 4.2 two different measures of efficiency: ratios of total run time t_T and ratios of average Lamb's coefficient iteration run time $\bar{t}_i$. The first provides an estimate of the real value of the GPU code as a whole by summarizing the true run time improvement between the two codes. Generally speaking, the GPU code is always faster, except for Case 1 where the CPU code runs nearly three times faster. This is because the problem is too small for the GPU's bandwidth to make up for its latency. We see a trend develop as the GPU code becomes progressively faster as the problem size grows. We can see from Table 4.2 that this trend becomes quite significant: the GPU code runs over 150 times faster than the CPU code for the largest case tested (Case 10). We recognize, however, that this metric fails to separate the differences in the numerical methods from those in the hardware. Specifically, we see that although the two codes require approximately the same number of Lamb's iterations to find a solution in the smaller cases, as the cases grow in both resolution and in the number of particles, the CPU code requires significantly more iterations for convergence.

We can obviously attribute some of the total run time improvement of the GPU code to this major difference, and therefore also compute the ratio for the average Lamb's coefficient iteration run time $\bar{t}_i$ (plotted in Figure 4.10b). This metric provides a fairer comparison between the two codes and concludes that the GPU code runs over 60 times faster than the CPU code for Case 10. We should be clear that this metric still includes some of the differences in numerical methods. To highlight one such

difference, notice that the run time per iteration actually decreases as the number of particles increases for a given simulation resolution. This can be attributed to our changing from using a fast Fourier solver in the CPU code to a conjugate gradient solver for the pressure-Poisson problem with Dirichlet pressure particle boundary conditions in the GPU code. We discuss this change in Section 4.3.4 but, put simply, the pressure-Poisson problem becomes easier to solve as more particles are added (e.g., the solver takes only 28% of the number of conjugate gradient iterations for Case 10G compared to Case 7G).

By considering the ratios of the run times for Cases 7 and 1 (both have one particle), we can attempt to normalize the effects of the differences in numerical methods and account for only the efficiency improvement attributed to the GPU hardware. Case 7 has 512 times as many discrete points as Case 1, and thus requires approximately 512 times as much work to solve. For the CPU code, Case 7 takes approximately 1260 times longer to solve each Lamb's coefficient iteration than Case 1. The same comparison for the GPU code reports that Case 7 takes only 14.6 times longer to solve each Lamb's coefficient iteration than Case 1. Not only do we see that the CPU code seems to require more than twice as much time as expected if it were to scale linearly with the problem size, but we find that the GPU code runs over 35 times faster than if it scaled linearly with problem size. Further, by taking the ratio of these two ratios, we see that the GPU hardware runs nearly 90 times faster than the CPU hardware. We believe that this comparison provides a clear measure of the

computational efficiency afforded by the hardware alone as the problem size increases by normalizing away the effects of the differences in the numerical methods between the two codes. This comparison also indicates the value of combining the GPU-centric code with traditional OpenMP/MPI parallelization methods, as the lowest-level computations performed on the GPU could run up to 90 times faster than on current OpenMP/MPI codes, with the potential to significantly improve the overall run time of the code as a whole.

Chapter 5

Validation

We have discussed in the preceding chapters a number of enhancements to the Physalis method, the component flow solver, and the collision model. Additionally, we have shown that our new GPU-centric implementation of these enhanced methods runs up to 90 times faster than legacy codes. In terms of accuracy, we find comparable results but with an increased convergence rate of Lamb's coefficient iterations and smaller particle force fluctuations. Here, we compile a wide array of validation tests to document the strengths and weaknesses of the present work.

In Section 5.1 we use a two-dimensional Taylor-Green vortex to prove that the background flow solver attains second-order accuracy. Section 5.2 returns to the work of G&P to illustrate the similarities and differences between the current and old methods in two key validation studies. In Section 5.3 we produce a comparison to the experimental work of Mordant and Pinton (2000) to serve as a benchmark of

Physalis in a more dynamic setting. Finally, we discuss a number of comparisons to various particle collision experiments in Section 5.4.

5.1 Flow solver validation

To illustrate the second-order convergence of the flow solver, we consider the two-dimensional Taylor-Green vortex (Chorin, 1967) that satisfies (1.1), namely,

$$u\left(x,y,t\right) = -\cos\frac{2\pi x}{L_x}\sin\frac{2\pi y}{L_y}e^{-2t/\tau}, \tag{5.1a}$$

$$v\left(x,y,t\right) = \sin\frac{2\pi x}{L_x}\cos\frac{2\pi y}{L_y}e^{-2t/\tau}, \tag{5.1b}$$

$$p\left(x,y,t\right) = -\frac{1}{4}\left(\cos\frac{4\pi x}{L_x} + \cos\frac{4\pi y}{L_y}\right)e^{-4t/\tau}, \tag{5.1c}$$

where $\tau = L_x L_y / \left(4\pi^2\nu\right)$ on the domain $0 \leq x/L_x, y/L_y \leq 1$. Table 5.1 summarizes the results of the study of a comparison of the error and convergence rate at time $t/\tau = 1$ in the manner described in Brown et al. (2001) using constant $C = 0.1$ for the determination of the time step using (2.8) for grids with resolution 32×32, 64×64, and 128×128. We confirm that the flow solver achieves second-order convergence in time and space as expected.

5.2 Physalis validation

To validate the numerous changes in the present work, we reproduce two tests of G&P and compare the accuracy of the two methods. With the exception of the

Variable	L_1-error			Rate
	32×32	64×64	128×128	
u	3.199×10^{-4}	4.006×10^{-5}	5.013×10^{-6}	2.00
v	3.199×10^{-4}	4.006×10^{-5}	5.013×10^{-6}	2.00
p	1.018×10^{-6}	1.296×10^{-7}	1.629×10^{-8}	1.99

Table 5.1: Error and convergence rates for Taylor-Green vortex flow solver validation test.

scalar product Lamb's coefficient calculations, the methods used in G&P mirror those of Z&P (in fact, they use the same code). By comparing against G&P, we effectively show the differences between the accuracy of the current work and its predecessors.

The first test considers the momentum conservation for the flow about a sphere in a cubic lattice by simulating a single particle located at the center of a triply-periodic domain. As explained in G&P, in the presence of a pressure gradient

$$\nabla p = P\hat{\mathbf{k}} + \nabla \hat{p}, \tag{5.2}$$

in which $\hat{p}$ is periodic and P is a constant, consideration of the momentum balance at steady state yields

$$F^* \equiv \frac{F_z}{(1-\beta)\, L^3 P} = 1, \tag{5.3}$$

where F_z is the force on the sphere, $\beta = \frac{4}{3}\pi a^3 L^{-3}$, and $L = 4a$ is the size of the domain. We reproduce a test in which we vary the truncation order l and particle resolution

$a/\Delta x$ using precisely the same parameters as in G&P, namely, $P^* = a^3 P \mu^{-1} \nu^{-1} = 10$, and $r/a = 1.25$ and plot the results in Figure 5.1.

The second test considers a similar situation, only now the sphere spins with a constant applied couple $L_{p,z}$. The work of G&P shows that, at steady state, the couple on the particle $L_{p,z}$ should equal the total hydrodynamic couple acting on the surface S of the simulation domain given by

$$L_{d,z} = \int_S d\mathbf{S}\ \mathbf{x} \times (\boldsymbol{\sigma} \cdot \mathbf{n}), \tag{5.4}$$

where $\sigma_{ij} = -p\delta_{ij} + \mu \left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right)$ is the stress tensor, so that the ratio $L^* \equiv L_{d,z}/L_{p,z}$ equals one. We plot the results in Figure 5.1 using the same simulation parameters as above and a rotation rate $\mathrm{Re}_\Omega = a^2 \Omega_z \nu^{-1} = 20$. For both cases, the results of G&P were generated using $a/\Delta x = 8$.

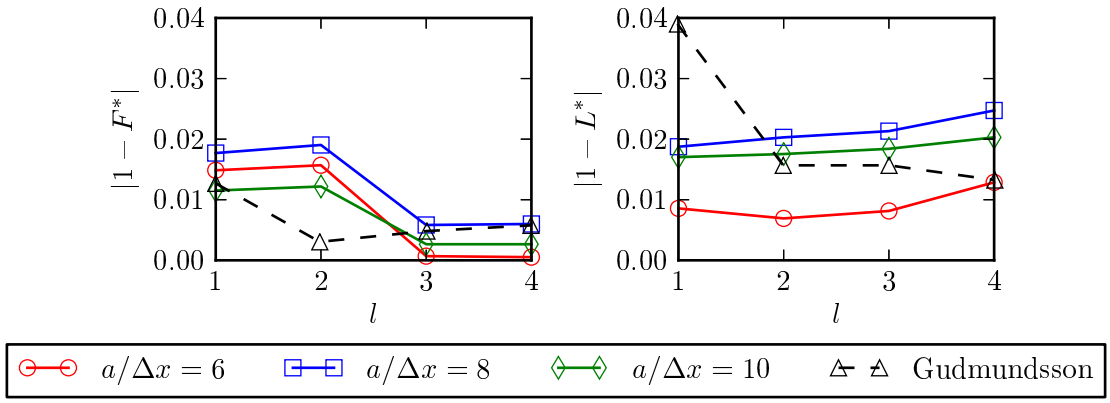


Figure 5.1: Results of the momentum conservation tests in G&P with $r/a = 1.25$, $P^* = 10$, and $\mathrm{Re}_\Omega = 20$. The tests of G&P use $a/\Delta x = 8$.

Judging the results illustrated in Figure 5.1, we can make some broad generaliza-

tions. For truncation of Lamb's series solutions up to $l = 1$ and $l = 2$, the current work exhibits approximately 50% larger error than G&P for the linear momentum conservation test, but consistently beats G&P by more than 10% for the angular momentum conservation test. For $l = 3$, however, with all $a/\Delta x$ tested for both linear and angular momentum conservation, the current work performs better than G&P. For $l = 4$, we see good results for both tests, which are essentially indistinguishable from those for $l = 3$.

The lack of monotonicity with increasing $a/\Delta x$ can be understood by noting that the position of the nodes used to interpolate the field values on the scalar-product integration surface will change with $a/\Delta x$. It may well happen that, for a particular particle position, the mutual relationship between the integration surface and the nodes is more favorable for one $a/\Delta x$ than for others. This indeed appears to be the case for the three values of $a/\Delta x$ considered here. For both $a/\Delta x = 6$ and $a/\Delta x = 10$, the integration surface generally cuts through the center of the Cartesian grid cells while it aligns with the cell edges for $a/\Delta x = 8$, representing the two extreme possibilities. It appears that the former pair of $a/\Delta x$ may be more directly compared while the latter takes on a different character. These considerations illustrate the connection between the position of the quadrature points, which is itself dependent on the radius of the cage, and the underlying Cartesian grid. This is evidently a matter of some complexity, which must be left for future work.

For additional commentary regarding particle resolution, refer to Section 5.3,

which shows that larger values of $a/\Delta x$ decrease force fluctuations encountered when particles move with respect to the grid.

5.3 Physalis benchmark: Sedimentation

The work of Mordant and Pinton (2000) presents high-quality data from experiments involving solid spheres of various sizes and densities sedimenting from rest in a quiescent fluid. Though much of the data is for high particle Reynolds numbers that would require very large grids, one experiment, for which $\mathrm{Re}_p = 41$, is an excellent benchmark against which to test the performance of Physalis for a moving particle.

Case #1 in Mordant and Pinton (2000) provides data for velocity as a function of time for the sedimentation of a glass sphere of radius $a = 0.25$ mm and density $\rho_p = 2560$ kg/m^3 in water at 25 °C. Using an acoustic Doppler shift method, Mordant and Pinton (2000) reports a terminal velocity $V_1 = 0.0741$ m/s ($\mathrm{Re}_p = 41$) and variance of 0.4 mm/s, which corresponds to an experimental uncertainty of 0.6%. The particle Galileo number is $\mathrm{Ga} = \nu^{-1}\left(|\rho_p/\rho_f - 1|\, 8a^3 g\right)^{1/2} = 49$.

To simulate this experiment we use a domain with cross Section $20a \times 20a$ with no-slip walls and height $48a$ with Neumann boundary conditions on the top and bottom; the results with Neumann lateral walls were the same. We perform simulations using $r/a = 1.2$ for the integration surface and $C = 0.5$ for the CFL number, with combinations of $a/\Delta x$ and l as summarized in Table 5.2. A plot of velocity as a func-

tion of time for each validation test may be found in Figure 5.2 (we have removed $t \lesssim 5$ ms in the experimental data due to a deficiency in the experimental method over this time period explained in Mordant and Pinton (2000)).

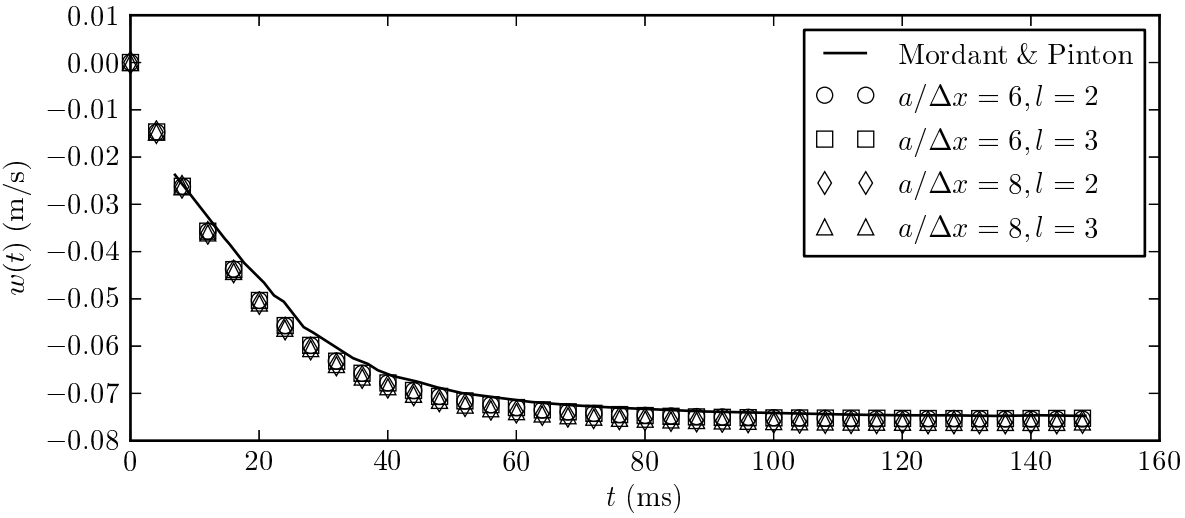


Figure 5.2: Benchmarking Physalis using Mordant and Pinton (2000), Case #1. Note that we have removed $t \lesssim 5$ ms in the experimental data due to a deficiency in the experimental method over this time period explained by Mordant and Pinton (2000).

Analyzing Figure 5.2 in detail, we see that all simulations produce satisfactory results while consistently, if slightly, overestimating the falling velocity (in magnitude). For all simulations, the settling velocity in the time range of approximately $10 < t < 50$ differs from the experimental result up to nearly 10%, but then converges to within 2.5% accuracy once terminal velocity has been reached. The work of Yang and Stern (2015) summarizes the results of several other numerical investigations all of which exhibit differences from the data similar in magnitude and direction as our

own.

Symbol	$a/\Delta x$	l	V_1	% error
——	Experimental		0.0741	0.6
○ ○	6	2	0.0746	0.675
□ □	6	3	0.0745	0.580
◊ ◊	8	2	0.0759	2.43
△ △	8	3	0.0759	2.43

Table 5.2: The results of our benchmarking study for the sedimentation of a single particle in quiescent flow. The experimental data is from Mordant and Pinton (2000), Case #1.

Table 5.2 quantifies the accuracy of the various simulations at terminal velocity where we can see that the higher resolutions produce an error at terminal velocity of 2.43% while the lower resolution simulations achieve results of 0.675% and 0.580%, which are very near the uncertainty associated with the experiments (0.6%). Thus, as already found in Section 5.2, the simulations with lower resolution ($a/\Delta x = 6$) produce results better aligned with experiment than those with higher resolution ($a/\Delta x = 8$). However, in contrast to the results of Section 5.2, the differences between using truncation order $l = 2$ and $l = 3$ appear to be quite minor.

In addition to the settling velocity, we consider in Figure 5.3 the three components of the hydrodynamic force on the particle normalized by the effective buoyancy force

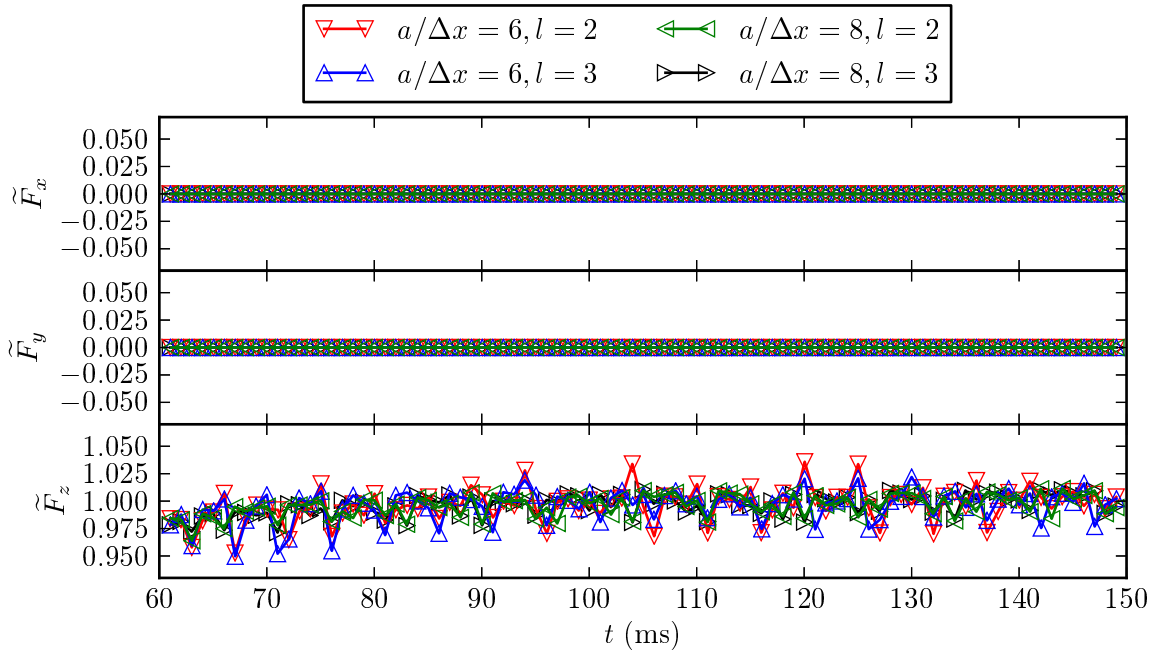


Figure 5.3: Force fluctuations in the Physalis benchmarking results for Mordant and Pinton (2000), Case #1.

as:

$$\widetilde{\mathbf{F}} = \frac{\mathbf{F}}{\frac{4}{3}\pi a^3 \left(\rho_p - \rho_f\right) g}. \tag{5.5}$$

Here, we can see how the motion of the particle against the fixed background grid (and the changing shape of the cage) affect the force fluctuations. The plots of $\widetilde{F}_x$ and $\widetilde{F}_y$ show that the fluctuations remain very small ($O\left(10^{-14}\right)$) and that $\widetilde{F}_z$ decreases from about 4% of the net gravitational force for $a/\Delta x = 6$ to 1% for $a/\Delta x = 8$. The results shown in Z&P for a particle sedimenting in a duct at $\mathrm{Re}_p \approx 22$ exhibit force fluctuations on the order of 4% for $a/\Delta x = 8$. Even though the Reynolds numbers of the two simulations differ (a circumstance which might be expected to reduce the

fluctuations as it enlarges the Stokes region), the comparison suggests that the force fluctuations in the present calculations are smaller, most likely because of the changes made to the cage construction described in Section 4.3.1, which enable the cage to change more smoothly as the particle moves.

5.4 Collision model validation

To test the validity of the collision model as a whole, we consider the results of several experimental data sets in the literature. The first comparison, testing normally-directed particle-wall collisions, reproduces the experimental results of Joseph et al. (2001), which summarizes the relationship between coefficient of restitution and Stokes number. To further illustrate the accuracy of the model in conjunction with the Physalis method as a whole, we reproduce an experiment of Gondret et al. (2002) and compare the position- and velocity-time trajectories of a series of normally-directed gravity driven particle-wall collisions. We consider the work of Yang and Hunt (2006) to show that our model also performs well in particle-particle collisions. Finally, we test our model's accuracy in oblique particle-wall collisions against the experiments of Joseph and Hunt (2004).

We contrast the present work to that of Sierakowski and Prosperetti (2016), which adopts an early version of the collision model discussed presently and concludes that the method requires increasingly large particle resolutions (up to 16 grid cells per

radius) as Stokes number increases. As formulated in the present paper, the method is no longer limited by such restrictions, and all tests have been performed with 8 grid cells per radius regardless of Stokes number.

5.4.1 Particle-wall collision

Normally-directed particle-wall collisions, the most extensively analyzed mode of collision, fall into two broad categories: film contact, where the wall is covered with only a thin layer of fluid Barnocky and Davis (1988); Davis et al. (2002), and immersed contact, where the impacting particle and wall are entirely contained within the fluid for the duration of the experiment. Interested in the latter category, we first consider the experimental results of Joseph et al. (2001), in which spherical particles of varying materials and sizes swing as a pendulum into a wall. The experiment was designed so that the particle and wall contact at the bottom of its oscillatory path. By determining the velocity of the particle before and after contact from a high-speed video recording of the collision, the work summarizes the observed coefficient of restitution e as a function of Stokes number.

To reproduce this experiment computationally, we linearize the pendulum motion and accelerate the particle normally to the wall using a linear elastic force. We simulate a subset of the experiment, using a glass particle of radius $a = 1.5$ mm and the following material properties: $\rho_p = 2540$ kg/m^3, $E = 0.6$ MPa, $\sigma = 0.23$, $e_{\text{dry}} = 0.98$, and $\mu_f = 0.5$. The fluid has the properties of water at 25°C. In a domain

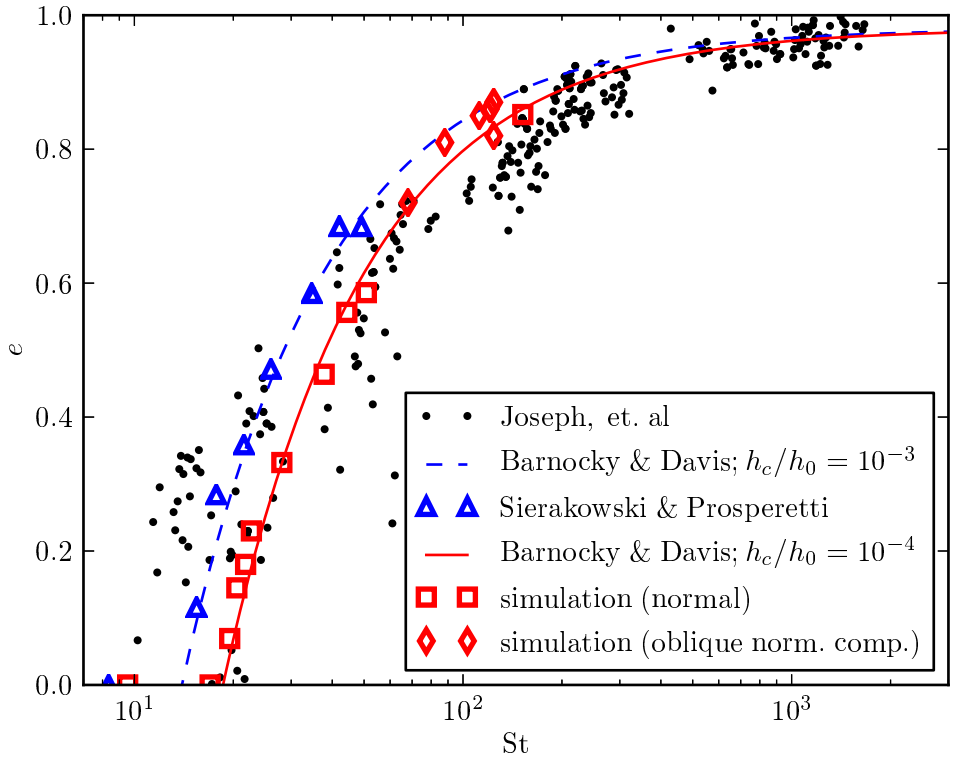


Figure 5.4: Comparison to the experimental normally-directed particle-wall collision results of Joseph et al. (2001).

of size $8a \times 8a \times 8a$ with a no-slip wall made of the same material as the particle, a Neumann condition at the opposing domain boundary, and periodic conditions elsewhere, we release the particle from a distance of 8.5 mm from the wall and vary the driving spring stiffness to change St.

We compute St and e from each simulation and plot the results in Figure 5.4 alongside the experimental data of Joseph et al. (2001), which compares favorably with the experimental data. Figure 5.4 plots two lines in addition to the data: the

curves defined by (3.24) for two different values of h_c/h_0, namely 10^{-3}—which was posed by Joseph et al. (2001) and adopted in Sierakowski and Prosperetti (2016)—and 10^{-4}, which we use in this work because it generates results better in line with those published in the experimental literature for all of our tests.

Accurate modeling of the relationship between e and St as plotted in Figure 5.4 is vital, but the experimental results indicate some strong variability in this measure, especially for $\mathrm{St} \lesssim 100$. As discussed in Joseph et al. (2001), the prevailing theory indicates that the surface roughness of the particle accounts for this variability, with rougher surfaces such as glass exhibiting greater variability than smoother surfaces like steel. Additionally, Figure 5.4 summarizes only the response near the wall and cannot validate the method as a whole, which is a marriage of the present collision model with the Physalis method.

We therefore consider the work of Gondret et al. (2002), which publishes the time history of the particle position and velocity in an experiment of a gravity-driven normal particle-wall collision. In this work, the particle is released from a sufficient height above the wall so that it travels at terminal velocity at the moment of contact. The particular experiment that we reproduce involves a steel sphere with $a = 1.5$ mm, $\rho_p = 7800$ kg/m^3, $E = 2.4$ MPa, $\sigma = 0.30$, $e_{\mathrm{dry}} = 0.97$, and $\mu_f = 0.1$. The fluid is silicone oil RV10, with $\rho_f = 935$ kg/m^3 and $\nu = 1.07 \times 10^{-5}$s/m^2. The presence of the wall into which the particle will collide at the bottom of the domain influences the particle terminal velocity, even when the particle begins far away. To provide the

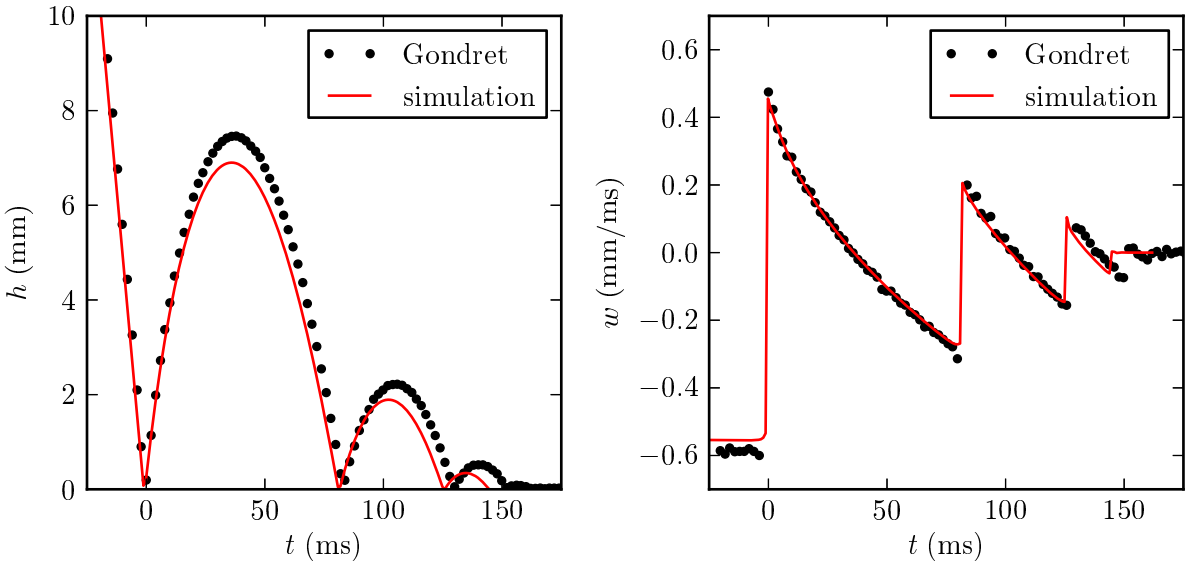


Figure 5.5: Comparison to the experimental particle-wall collision results of Gondret et al. (2002).

particle sufficient space to reach terminal velocity with a fully-developed wake before contact, we initialize the particle with its terminal velocity $w = -0.58$ m/s at a height of 76 mm, which requires a domain of size $52a \times 14a \times 14a$, with the longest dimension in the gravitational direction and Neumann boundary conditions everywhere except for the no-slip bottom. For the initial collision, we have $\mathrm{Re}_p = 156$ and $\mathrm{St} = 144$, which we compare to the experimental values of $\mathrm{Re}_p = 165$ and $\mathrm{St} = 152$, which represents an error of 5.3% in the terminal velocity of our simulation. We have performed this simulation with higher particle resolution but found a very similar result. Performing the simulation with no bottom results in a terminal $\mathrm{Re}_p = 159$,

leading us to believe that the large (but still limited) domain size continues to hinder the terminal velocity that we may achieve in this test.

Considering this discrepancy at the initiation of the collision, we plot the results of the simulation and the experiment in Figure 5.5. We see, generally speaking, a favorable result, especially in the plot of the particle velocity. As position is an integrated quantity, a small error at the onset of contact will grow over the course of the trajectory, which is exhibited in the plot of particle position. Should the particle have initiated contact marginally faster, it is clear that the particle position plot would have matched slightly better, and we conclude that the collision model performed well, but that the overall accuracy of the simulation was limited by the domain size.

In Figure 5.6, we draw specific attention to the importance of the flow environment experienced by the particle during this collision. The momentum of the fluid entrained by the particle during its approach strongly influences the trajectory of the particle after contact, the resolution of which which accounts for a significant portion of the accuracy of the results plotted in Figure 5.5. Further, even at this relatively high Stokes number, we find the modeled lubrication forces of Section 3.1 to be extremely important. Figure 5.7 plots the results of a simulation identical to the normal particle-wall contact just described except for the absence of lubrication modeling. Without including the lubrication model, the particle approaches the wall with velocity too high because we fail to resolve the large viscous dissipation associated with fluid being

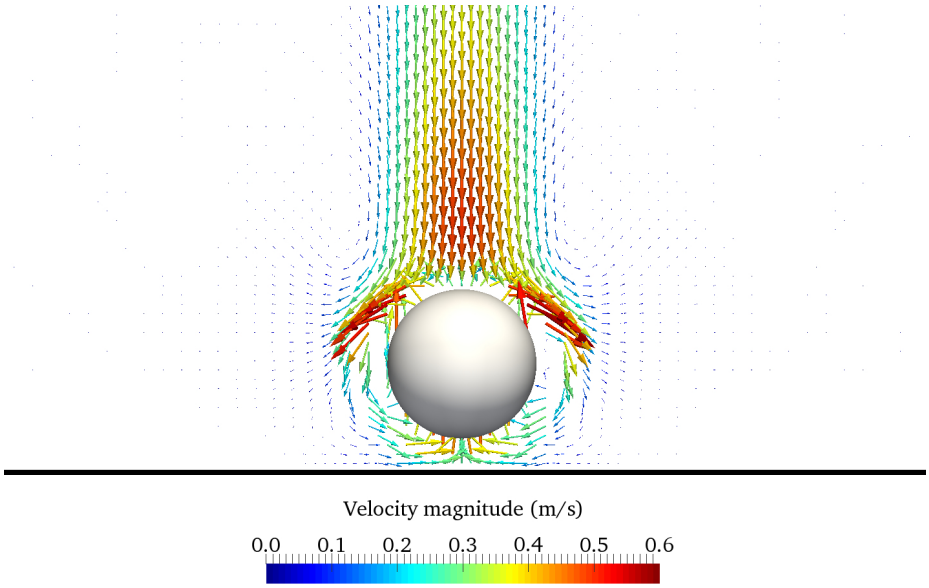


Figure 5.6: Rebound velocity field (2 ms after contact). The particle is moving upward with velocity $w = 0.45$ m/s.

squeezed from the gap between the particle and wall at small separation. Next, the particle completes the contact with a very similar coefficient of restitution as we do not expect e to change drastically for a small percentage increase in St ≈ 150 (see Figure 5.4). It finally rebounds with velocity too high as we again fail to resolve the viscous dissipation associated with fluid being pulled back into the gap between the particle and wall.

As we mention in Section 3.2, there is some concern as to the effect of decreasing the Young's modulus E of the particle. Though Joseph et al. (2001) indicates that the coefficient of restitution is little influenced by the value of E, Kempe and Fröhlich (2012b) shows that decreased values of E lead to contact events stretching for sig-

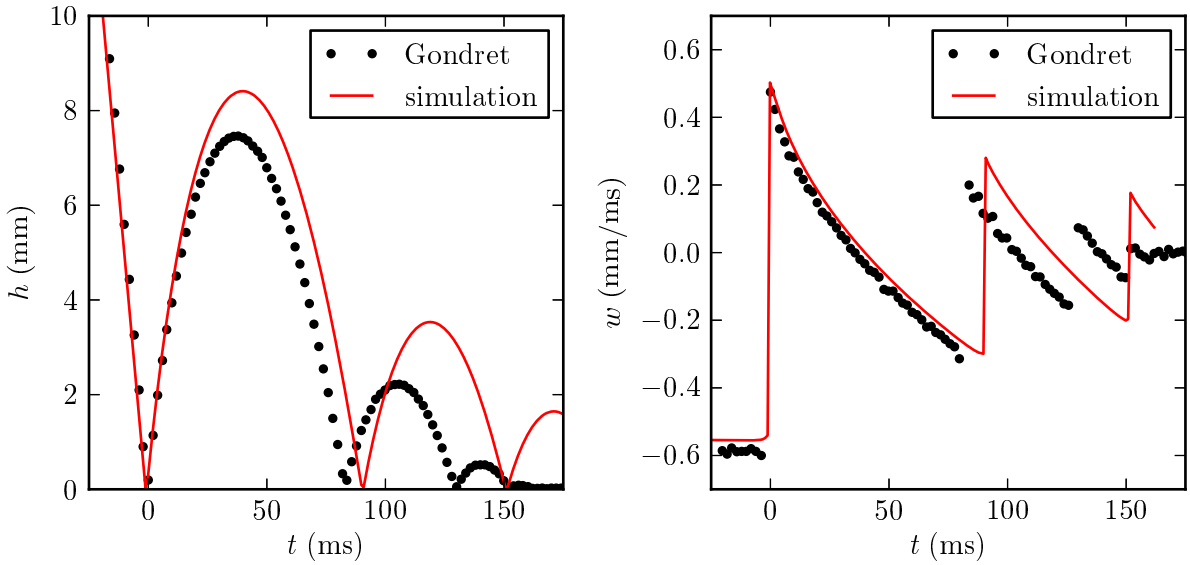


Figure 5.7: Comparison to the experimental particle-wall collision results of Gondret et al. (2002), but without the lubrication model.

nificantly longer time than is physically accurate. For example, in the experiment of Gondret et al. (2002) simulated numerically above, Kempe and Fröhlich (2012b) indicates that the Hertz contact theory suggests the contact should be complete in 0.012 ms. In contrast, the contact in our simulation takes 0.44 ms to complete, which is 37 times longer than expected. Matching to the same order of magnitude, this duration of contact compares favorably with that achieved by the contact model of Kempe and Fröhlich (2012b). The two models differ in the number of fluid time steps used to resolve the contact, however. While Kempe and Fröhlich (2012b) enforces that the contact occur over a fixed number of time steps (typically, 20) by choosing

k_n and η_n in (3.12) by solving the nonlinear differential equation (3.19) for every contact, we choose k_n and η_n as described in Section 3.2 and allow the simulation to evolve using an adaptive time step determined by the fluid velocity field to maintain a specified CFL number (see, e.g. Sierakowski and Prosperetti, 2016). During contact, the flow conditions require an average time step of duration 0.0043 ms, which is much smaller than the average time step of 0.11 ms used over the entirety of the simulation. Due to the short duration of the time steps—required by the Physalis method to appropriately resolve the impulse applied to the fluid—we took 112 time steps to resolve the contact. Further, the highly dynamic nature of the contact increased the average number of convergence iterations from 3.9 iterations per time step for the simulation as a whole to 5.4 iterations per time step during contact.

This behavior may be seen as both a benefit and a drawback as we appropriately resolve the fluid time scale associated with the contact impulse but require numerous small time steps to do so. Note that it would require an order of magnitude more time steps to appropriately resolve the time scale associated with the stiffness E of the solid particle. It is clear that, in the case of frequent contacts at large St, the present method will frequently suffer from decreased time step size and increased number of Lamb's iterations.

While keeping all of the above analysis in mind, we must draw attention to the fact that we are comparing the trajectory of our simulation against a single experimental result. Judging from Figure 5.4, we could expect $0.7 \lesssim e \lesssim 0.9$ for a normal particle-

wall collision at $St \approx 150$, indicating a relatively wide range of experimental responses. We observe that our $e = 0.85$ lies towards the top of this range and also that the particular experimental result against which we are comparing experiences an even larger rebound, indicating that we may expect the average case to exhibit a lower rebound better aligned with the result of our simulation. From this perspective, without significantly more extensive statistical understanding of the experimental results—data that has not been published in the literature, if it exists at all—we may reasonably conclude that our model performs adequately.

5.4.2 Particle-particle collision

The work of Yang and Hunt (2006) extends that of Joseph et al. (2001) to particle-particle collisions using a similar pendulum methodology by swinging a sphere into a second identical sphere held stationary. To validate our collision model for normal particle-particle contact, we reproduce the experiment using glass spheres of radius $a = 6.35$ mm and material properties specified above. The fluid, in a domain of size $8a \times 8a \times 16a$ with Neumann conditions on all faces, is water at 25° C. Again, we linearize the pendulum motion and accelerate the particle towards contact at varying Stokes number by modulating the driving linear elastic spring stiffness. The results, plotted along with the experimental results in Figure 5.8, match the expected behavior well.

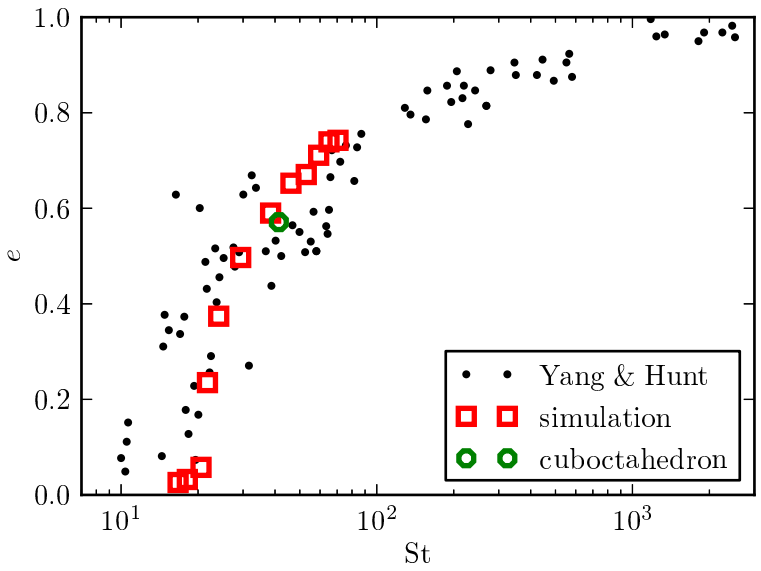


Figure 5.8: Comparison to the experimental particle-particle collision results of Yang and Hunt (2006).

5.4.3 Oblique wall collision

Normally-directed collisions comprise but an exception to the rule, as a typical collision may occur at any angle. The literature (Joseph and Hunt, 2004; Yang and Hunt, 2006) indicates that, during an oblique collision, the normal component follows the same rule as in the purely normally-directed contact tested above, and we have added the coefficients of restitution resulting from the normal component of oblique contact (i.e., "oblique norm. comp." in the legend) to Figure 5.4 to illustrate this effect. To investigate the effect of the tangential component, it is common to define the normalized incident and rebound angles Ψ_0 and Ψ_1, respectively, as

$$\Psi_0 = \frac{w_0^t}{w_0^n}; \tag{5.6a}$$

$$\Psi_1 = \frac{w_1^t}{w_0^n}, \tag{5.6b}$$

where the t and n superscripts refer to the components of velocity tangential and normal to the wall and the 0 and 1 subscripts indicate velocity before and after contact, as usual.

We perform one final test of our model, where we compare against the experimental results of Joseph and Hunt (2004) for oblique particle-wall collisions. This experiment again uses a pendulum to generate the collision, tilting the contact angle with the wall in a plane so as to not introduce the influence of gravity at the point of contact. The experiments were performed for both steel and glass particles at various

Stokes numbers, and the results are plotted in Figure 5.9 for St < 1000. Interestingly, Joseph and Hunt (2004) observed a recoil effect for the glass particles, an effect that they attributed to the larger surface roughness of that material ($O\,(0.1\ \mu\text{m})$ for glass compared to $O\,(0.02\ \mu\text{m})$ for steel). This theory is supported by the observation that steel particles also exhibited this behavior for very large St, when fluid viscous forces play a smaller role.

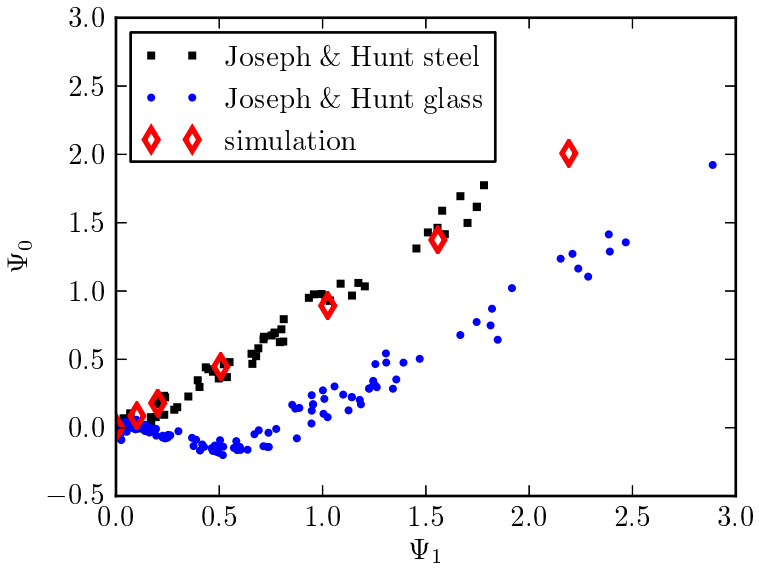


Figure 5.9: Comparison the experimental oblique wall collision results of Joseph and Hunt (2004) for particles of steel and glass. Only results with St < 1000 are plotted here.

To reproduce these experiments, we accelerate a particle with radius $a = 6.35$ mm towards the wall with a linear spring at varying angles of incidence using the material

properties for both steel and glass as defined above. We use a domain of water at 25° C of size $8a \times 16a \times 8a$ with the longest direction parallel to the no-slip wall, a Neumann boundary on the opposing side, and periodic boundaries elsewhere. The results of the simulations are plotted in Figure 5.9 for steel, and the results match well. Unfortunately, the simplified tangential contact model that we have adopted from Simeonov and Calantoni (2012) is unable to capture the recoil effect observed for the rougher glass, presumably because it lacks a modeled solid material damping. We hope to revisit this effect in future work.

Chapter 6

Applications

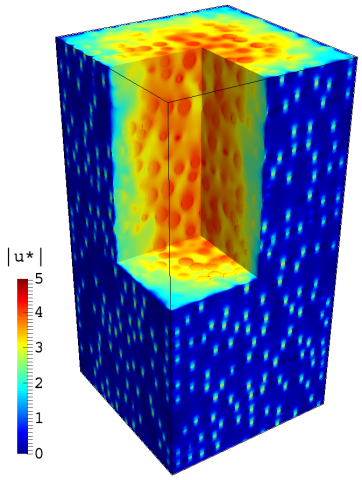
The purpose of this chapter is to briefly summarize some early results and simulations that have been performed using the present work. We aim to provide some basic understanding of the capabilities of the code, including the amount of time required to run each of the simulations. All of the simulations presented here were performed on either a single Nvidia GeForce GTX Titan GPU with an Intel Xeon E-5645 CPU or a single Nvidia Tesla K40 GPU with an Intel Xeon E5-2650v2 CPU. Since many of these simulations were performed with older versions of the code, which has been steadily improving in accuracy, stability, and efficiency, the run time and iteration statistics listed should be taken as an upper bound.

6.1 2048 particles sedimenting in a duct

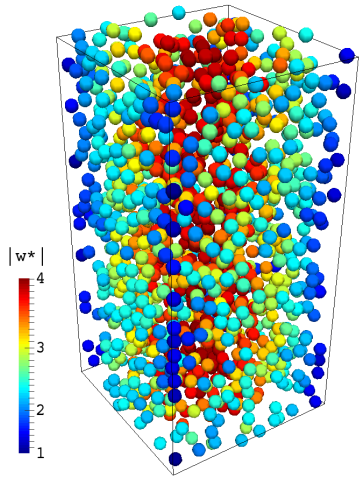
We present a sizeable simulation involving 2048 particles sedimenting in a vertically periodic duct with an average volume fraction of 20% (for triply-periodic sedimentation, see Section 6.5. The particles and fluid use the properties of Section 5.3, which provides the single-particle terminal velocity $V_1 = 0.0741$ m/s and characteristic time $\tau_{95} = 55$ ms (see Mordant and Pinton (2000)) with which to normalize the fluid and particle sedimentation velocities produced by the simulation as $u^* = u/V_1$ and $w^* = w/V_1$, respectively, as well as the time $t^* = t/\tau_{95}$. We use simulation parameters $a/\Delta x = 8$, $l = 3$, and CFL constant in (2.8) $C = 0.5$. The domain size is $28a \times 28a \times 56a$ with four no-slip lateral walls and periodic boundary conditions in the longest (gravity) direction. The particles have a Galileo number $\mathrm{Ga} = \nu^{-1} \left(|\rho_p/\rho_f - 1| \, 8a^3 g \right)^{1/2} = 49$.

Figure 6.1 illustrates the magnitude of the normalized fluid and particle velocities, $|u^*|$ and $|w^*|$, respectively, at $t^* = 2.4$ and Figure 6.2 shows $\langle j^* \rangle_z$, the volumetric flux averaged over the gravity direction, at $t^* = 3.5$. In Figure 6.3, we plot the mean absolute (i.e. not relative to the fluid) velocity $\overline{|w^*|}$ plus/minus the standard deviation σ_{w^*}, and maximum/minimum particle velocities $|w^*|_{M/m}$ as functions of time. In Figure 6.4 we plot the vertically-averaged particle volume fraction $\langle \beta \rangle_z$ over the cross section of the duct at times $t^* = 0$ and $t^* = 3.5$.

Even though, due to periodicity, the number of particles is effectively infinite, their velocity does not continue to increase because of the dissipation caused by



(a) Visualization of the fluid velocity. The particles are not shown for clarity.



(b) Visualization of the particle velocity. The fluid is not shown for clarity.

Figure 6.1: 2048 particles sedimenting under gravity in a periodic duct at $t^* = t/\tau_{95} = 2.4$. Here, $|u^*|$ and $|w^*|$ are the fluid and the particle velocities normalized by the single-particle terminal velocity as determined in Section 5.3.

the walls. However, we notice in Figure 6.4 the accumulation of particles along the walls, which is reminiscent of phenomena shown in Shaffer and Gopalan (2013). This phenomenon is explained by the fact that, when a fast particle in the core of the duct collides with a slower-moving particle and pushes it toward the wall, the effective restitution coefficient is too weak to push the particle back into a region of significant flow velocity. It is somewhat surprising that the minimum velocity shown in Figure 6.3, which is typically due to particles falling near the walls, is as large as the single-

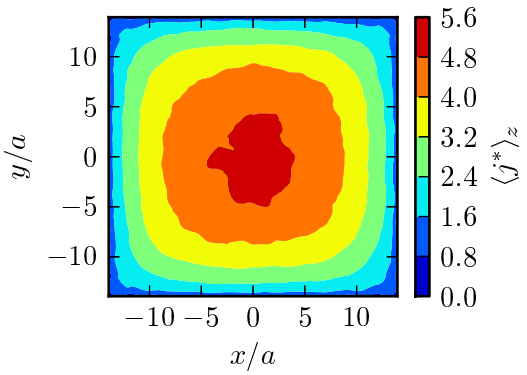


Figure 6.2: The volumetric flux averaged over the gravitational direction at $t^* = 3.5$.

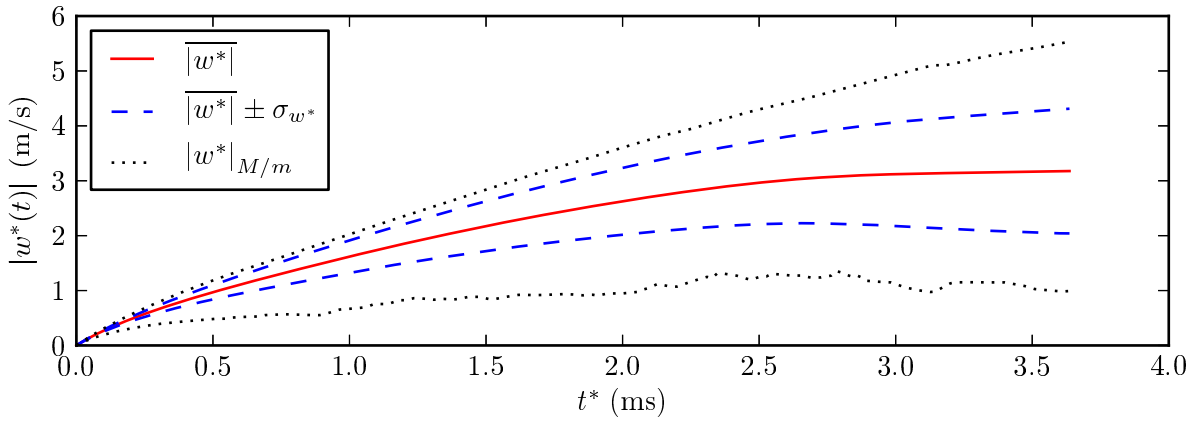
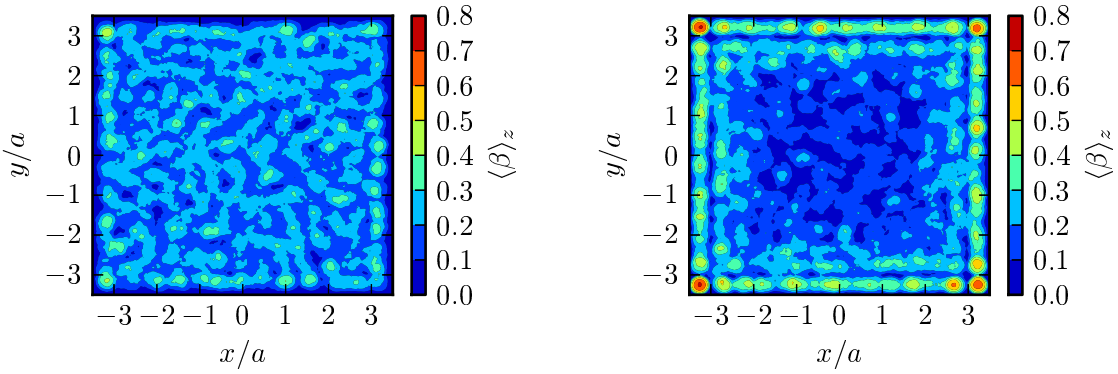


Figure 6.3: The particle velocity $w^* = w/V_1$ over time with mean $\overline{|w^*|}$ plus/minus standard deviation σ_{w^*}, and maximum/minimum $|w^*|_{M/m}$.

particle terminal velocity. This effect is due the strong shear caused by the flow in the core of the duct, where we observe particle velocities as large as five times the single-particle terminal velocity.

This simulation required about four weeks of computation time for over 10,500

time steps averaging 12 Lamb's coefficient iterations per time step. We attempted this simulation also with a coarser discretization, $a/\Delta x = 6$, but at some point Lamb's coefficients failed to converge due to insufficiently resolved collisions.



(a) $t^* = 0$ (b) $t^* = 3.5$

Figure 6.4: Averages of particle volume fraction $\langle\beta\rangle_z$ over the gravitational direction.

6.2 Moderate Reynolds number

To illustrate that Physalis is not by any means limited to regimes of small Reynolds number, we illustrate in Figure 6.5 a particle with a dynamically shedding wake at $\mathrm{Re}_p = 600$. This simulation, with a stationary particle of radius $a = 1$ a distance of $3a$ away from the inflow (all other walls are Neumann), resolves the particle with 17 cells per radius, $r_s/a = 1.1$, and $l_{\max} = 3$ in order to capture the thin layer in which the Stokes approximation is valid. The domain occupied $18a \times 6a \times 6a$ with resolution

$300 \times 100 \times 100$ and used $\rho_f = \nu = 1$. In May 2013 (very early in the development of the code), the simulation ran for 12 days, taking 103,400 time steps.

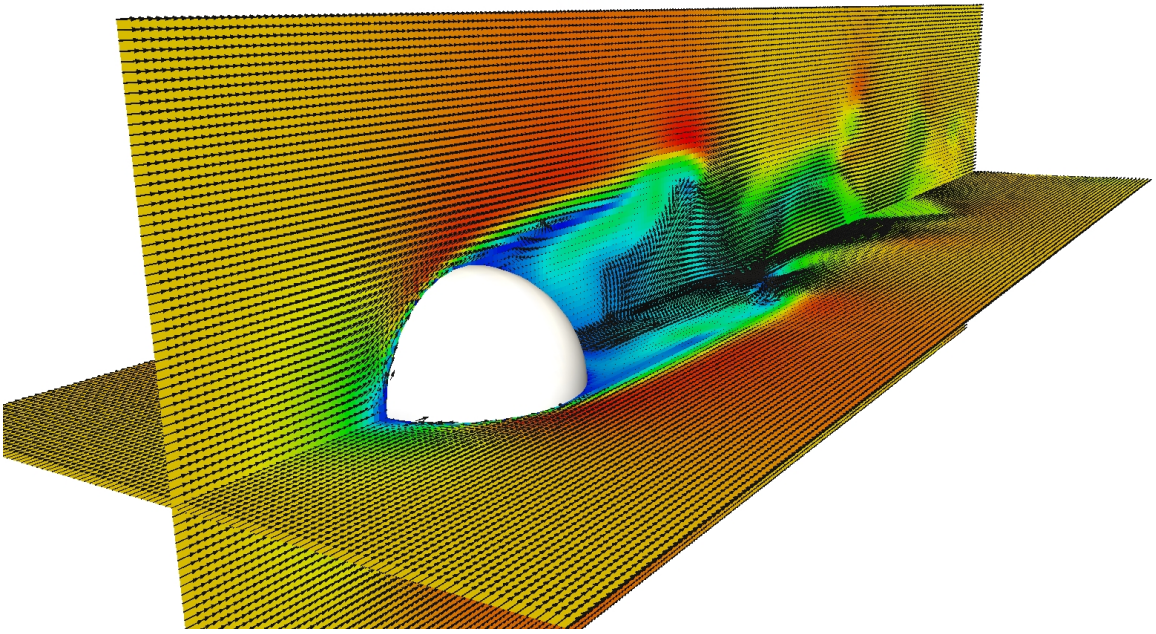


Figure 6.5: Flow around a sphere at $\mathrm{Re}_p = 600$. Note the fine resolution required to capture the thin layer near the particle in which the Stokes approximation is valid.

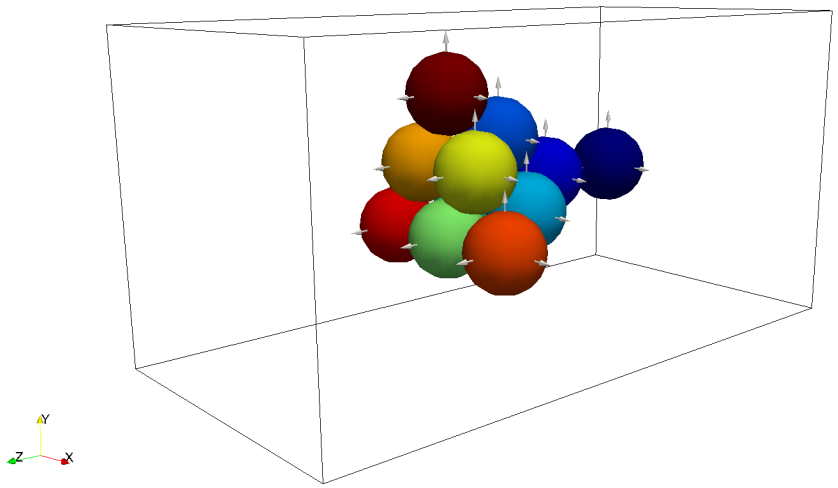
6.3 Three-dimensional billiards

Often used for testing the contact model, we illustrate the concept of the three-dimensional billiards simulation in Figure 6.6. Eleven particles with radius $a = 1$ are allowed to move freely about a periodic domain of size $4 \times 4 \times 8$ with resolution $64 \times 64 \times 128$ with initial positions summarized in Table 6.1. Typically, we test with $\rho_p = \rho_f = \nu = 1$. As a test case, the simulation runs quite rapidly, with useful data

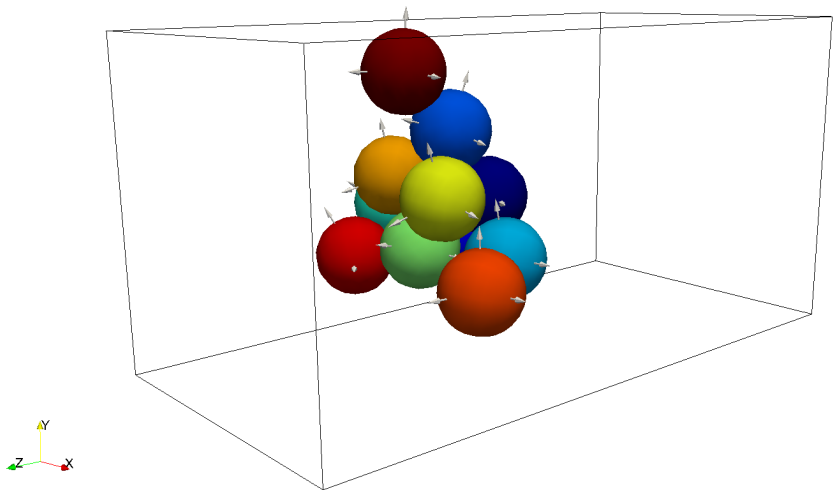
within 30 minutes. Of course, for rapid testing, shrinking the domain and cutting the resolution in half will also often provide valuable information.

N	X	Y	Z
0	0	0	-2
1	0	0	$-\frac{\sqrt{6}}{3}$
2	0	$\frac{\sqrt{3}}{3}$	0
3	$\frac{1}{2}$	$-\frac{\sqrt{3}}{6}$	0
4	$-\frac{1}{2}$	$-\frac{\sqrt{3}}{6}$	0
5	0	$-\frac{\sqrt{3}}{3}$	$\frac{\sqrt{6}}{3}$
6	$\frac{1}{2}$	$\frac{\sqrt{3}}{6}$	$\frac{\sqrt{6}}{3}$
7	$-\frac{1}{2}$	$\frac{\sqrt{3}}{6}$	$\frac{\sqrt{6}}{3}$
8	1	$-\frac{\sqrt{3}}{3}$	$\frac{\sqrt{6}}{3}$
9	-1	$-\frac{\sqrt{3}}{3}$	$\frac{\sqrt{6}}{3}$
10	0	$\frac{2\sqrt{3}}{3}$	$\frac{\sqrt{6}}{3}$

Table 6.1: The initial arrangement of particles in the three-dimensional billiards test simulation. Particle 0 is pushed into the others with a constantly applied force $F_z = 500$.



(a) Initial configuration



(b) Collision

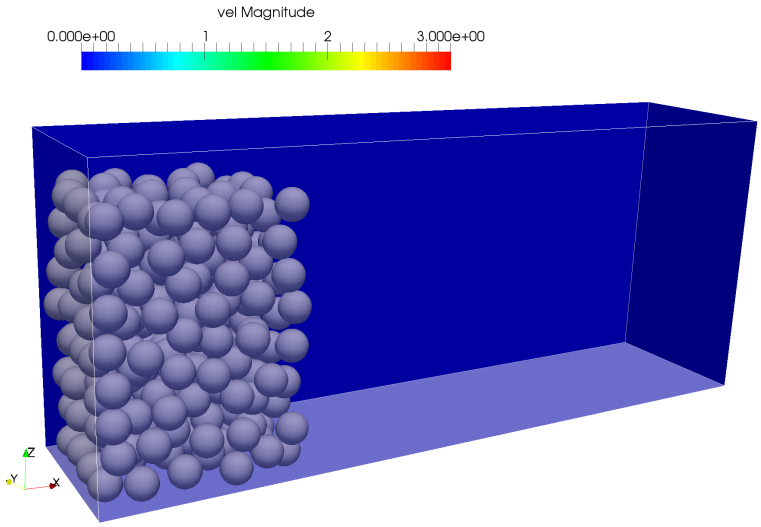
Figure 6.6: The three dimensional billiards test case. The arrows represent basis vectors used to track the orientation of the particles.

6.4 Collapsing particle column

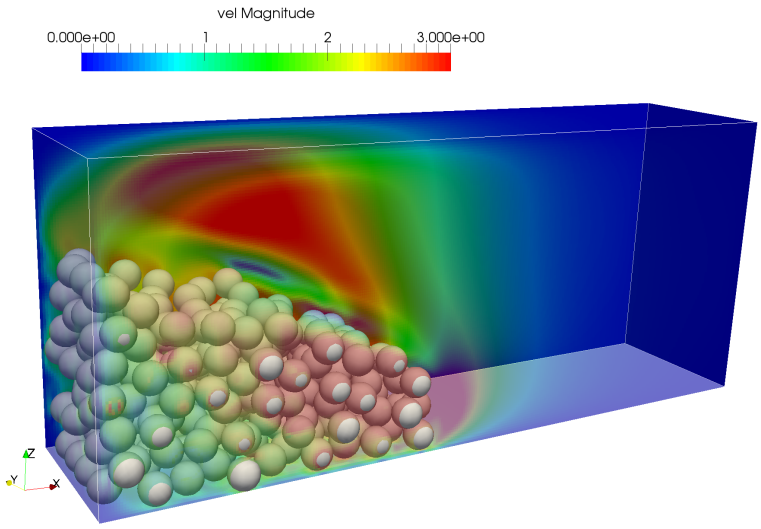
In an attempt to begin considering phenomena like turbidity currents, we consider the simulation illustrated in Figure 6.7, where 240 particles with $a = 1$ and $\rho_p = 3$ are randomly placed in a column as pictured in Figure 6.7a. As the column is released into the fluid with domain size $54a \times 11a \times 20a$ with resolution $324 \times 66 \times 120$ and bottom, left, top, and right no-slip walls (periodic elsewhere) with gravity pointing in the $-z$-direction, it collapses, entraining fluid along the way, as illustrated in Figure 6.7b. The simulation required 10 days to run 15,237 time steps, at which point the particles had settled into a bed on the bottom of the domain. A special acknowledgement is owed to Yayun Wang for her work on this simulation.

6.5 Periodic sedimentation

This setting was actually part of a parametric study of the effect of volume fraction and particle-fluid density ratio on particle sedimentation performed by Daniel Willen. In a periodic domain of size $20a \times 20a \times 60a$ with a resolution of $160 \times 160 \times 480$, we simulate the gravitational sedimentation of up to 2,000 glass particles in oil. The material properties are: $a = 2.1$ mm, $\rho_p = 0.00289$ g/mm^3, $E = 0.65$ MPa, $\sigma = 0.5$, $e_{\text{dry}} = 0.98$, $\rho_f = 8.75 \times 10^{-4}$ g/mm^3, and $\nu = 0.0175$ mm^2/ms (it is best to work in the gram, millimeter, millisecond scale when simulating real material properties). These simulations generate data at a rate of 8 days per one second of simulated time,



(a) Initial configuration



(b) Collapsing bed

Figure 6.7: The collapse of a column of 240 particles.

taking approximately 12,000 time steps at 4 Lamb's iterations per time step.

As the particles fall under gravity with no stationary body to dissipate energy, the sedimenting cloud and its entrained fluid will accelerate downward without bound. To prevent this from happening—essentially fixing our reference frame to that of the mean sedimentation velocity of all of the particles—we apply an adverse pressure gradient to balance the weight of the particles. Since the simulation cannot be perfectly accurate, we use a PID controller to balance this pressure gradient subject to the constraint that the mean particle acceleration is zero.

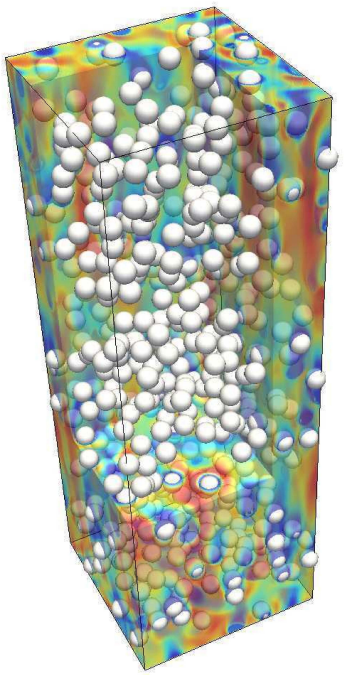


Figure 6.8: The periodic sedimentation of 500 particles.

6.6 2048 particles settling into a bed

Daniel Willen has also begun to investigate the settling of particles under gravity towards a no-slip wall. This simulation illustrates the capability of the contact models to support a bed of settled particles similar to the experimental work of Richardson and Zaki (1954). We randomly initialize 2048 glass particles with material properties given in the previous section with a volume fraction of approximately 31%. The fluid domain of size $24a \times 24a \times 60a$ with resolution $192 \times 192 \times 480$ and material properties has a no-slip bottom, Neumann top, and periodic walls. The particles require approximately one second of simulated time to settle to the bottom, which took nearly two months of simulation time, totalling 10,173 time steps. The final settled particle volume fraction is 63%. The biggest limitation here is the large number of Lamb's iterations required for convergence (≈ 25) when the particles are packed with very little fluid motion.

6.7 Fluidized bed

After achieving a settled bed, Daniel Willen fluidized the bed by injecting fluid upward against gravity. In this case, an initially settled bed of 128 particles with the same material properties as the settling simulation bed above is fluidized with an inflow velocity of 0.078 m/s. The domain is of size $12a \times 12a \times 20a$ with resolution $96 \times 96 \times 160$, and it requires 9 days to simulate two seconds of simulated time using

25,988 time steps with an average of 6 Lamb's iterations per time step.

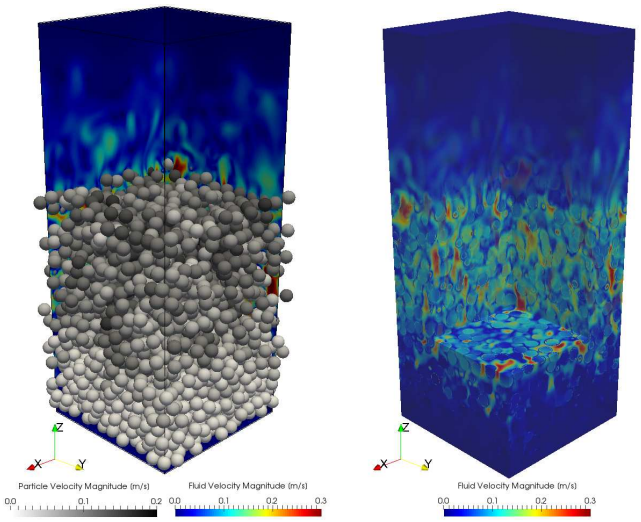


Figure 6.9: 2048 particles settling into a bed.

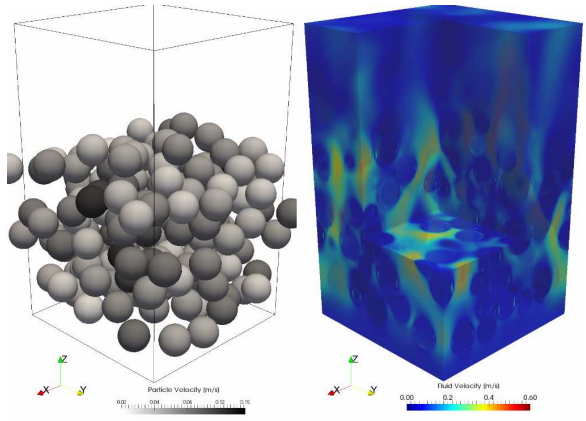


Figure 6.10: A small fluidized bed.

6.8 Johns Hopkins University logo

This example does an excellent job of illustrating how far our methods have progressed, especially with regard to numerical stability. Here, we have 229 stationary particles of radius $a = 1$ arranged in a crude approximation of the Johns Hopkins University logo in a pseudo-two-dimensional periodic domain of size $139.5a \times 34.5a \times 4$ with a resolution of $838 \times 208 \times 24$, which equates to six cells per radius particle resolution. A pressure gradient of $\mathbf{e}_x \cdot \nabla p = -50$ drives the flow in fluid with $\rho_f = \nu = 1$. The mean fluid velocity is approximately 60, with turbulent fluctuations up to 100 more, indicating that particles are temporarily exposed to local flow conditions approaching $\mathrm{Re}_p = 300$, with a Reynolds number based on the mean flow velocity and an estimate of the width of the logo near 3,000. The simulation ran for approximately 4 flow through times, taking 70,432 time steps with an average of 2.3 Lamb's iterations per time step in 13 days of computational time.

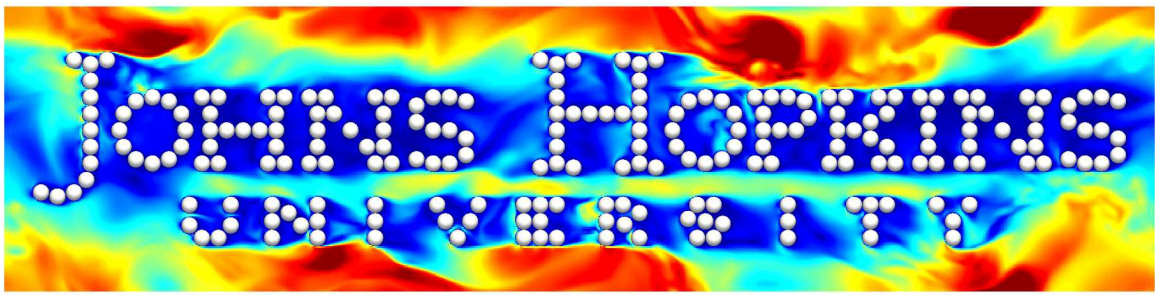


Figure 6.11: A crude approximation of the Johns Hopkins University logo in pressure-driven flow (left to right).

Chapter 7

Conclusions

Motivated by the limited understanding of the rich physics underlying disperse particle flows as described in Chapter 1, we conclude that the work published in this dissertation advances the state of disperse particle flow research by contributing a new tool for performing resolved-particle flow simulations on a GPU. We have put forward a number of enhancements to the Physialis method for introducing the influence of spherical particles to an incompressible viscous fluid in Chapter 2. For the first time, we have incorporated with Physalis a physically accurate collision model that captures the unresolved lubrication and contact effects in Chapter 3, and even propose a new nonlinearly damped Hertzian contact model. In Chapter 4, we documented some of the finer details of our GPU-centric implementation of Physalis that we have shown to run up to 90 times faster than a legacy implementation with significantly improved numerical stability. To prove the accuracy of this new tool,

we then presented in Chapter 5 a variety of validation and benchmarking simulations that compared favorably to experimental results published in the literature. Finally, we surveyed the breadth of applications that our tool is capable of simulating in Chapter 6.

We are excited by the results of this work, as it shows for the first time that Physalis is capable of simulating in a reasonable amount of time sufficiently many particles for limited statistical quantification of disperse particle flow phenomena. For example, by simulating thousands of particles, we are progressing towards the development of improved closure models for reduced-order approximations. One of the factors most strongly limiting the development of these closure models is the extreme size of the parametric space that describes the physics, including fluid-particle density ratios, particle momenta, particle concentrations, and relative particle positions, just to name a few. Mapping a space this large will require unprecedented amounts of data and, in the opinion of this author, the ability far beyond that afforded by the intuition of current researchers to find the most valuable correlations and trends in this data. Future researchers are not likely to bring with them very much improved intuition, but the computational capabilities afforded by recent advances in machine learning, especially the so-called deep neural networks (Kriegeskorte, 2015), are likely to play a very important part in teasing out subtle correlations in these tremendously complicated search spaces. This author envisions asking a deep neural network to predict which forces a point particle should experience by teaching it using the results

of particle-resolved simulations, for example.

At the same time, the present work confirms the relevance of GPUs in performing the simulations that generate the particle-resolved data in the first place. The author sees the present work as an elaborate proof of the concept that GPUs can help in the numerical simulation of disperse particle flows, with the ultimate goal of pulling together many GPUs spread across a distributed high-performance computer in order to increase the number of particles in a simulation to 100,000 and beyond. By utilizing GPUs as the lowest-level computational engine within the traditional domain decomposition methods employing both the Message Passing Interface (MPI) and Open Multi-Processing (OpenMP) application programming interfaces to communicate between processing nodes and GPUs, respectively. To achieve this goal, we have much to learn about how best to communicate between the various components of the domain, and solving the pressure-Poisson problem (2.4) in an efficient way will likely present one of the largest hurdles to be overcome. There exist many other questions of how precisely to proceed, including how best to pass particles between components of the domain. Thus, the step from an optimized single-GPU implementation to an optimized many-GPU implementation is far from automatic and will require significant algorithmic research and development. The marriage of these two ideas—namely many-GPU resolved-particle simulation for generating data and the application of deep neural networks (which also benefit tremendously from GPUs) for analyzing it—would help to significantly advance physical understanding of disperse

particle flows.

Of course, as technology continues to advance ever more rapidly, we must be careful to avoid tying ourselves too tightly to any one computing technology or technique, which includes even the relatively cutting-edge GPUs. We must keep our eyes open for future advancements in technology and be prepared to change course so that we may continue to lead the way in the application of these computational capabilities, which hold the keys to our understanding of the world around us.

Appendix A

Transformation to particle frame

In Section 2.3, we change from the laboratory frame of reference in which the domain containing the system is stationary to the frame of the particle. In the laboratory frame, the fluid velocity $\mathbf{U}$ and pressure p with conservative body force per unit mass are given by the incompressible Navier-Stokes equations 1.1, which we rewrite here using the material derivative

$$\frac{D\mathbf{U}}{Dt} = \frac{\partial \mathbf{U}}{\partial t} + (\mathbf{U} \cdot \nabla)\,\mathbf{U}, \tag{A.1}$$

as

$$\frac{D\mathbf{U}}{Dt} = -\frac{1}{\rho_f}\nabla p + \nu \nabla^2 \mathbf{U} + \mathbf{g} \tag{A.2a}$$

$$\nabla \cdot \mathbf{U} = 0. \tag{A.2b}$$

We reference positions within the fluid from the center of the particle using position vector

$$\mathbf{r} = r\mathbf{e}_r + \theta\mathbf{e}_\theta + \varphi\mathbf{e}_\varphi = r_1\mathbf{e}_1 + r_2\mathbf{e}_2 + r_3\mathbf{e}_3, \tag{A.3}$$

where $\mathbf{r}$ changes due to particle translation and rotation of the $\mathbf{e}_i$ fixed to it. The rate of change of $\mathbf{r}$ is therefore given by

$$\begin{aligned} \mathbf{r} &= r_i\mathbf{e}_i \\ \frac{D\mathbf{r}}{Dt} &= \frac{D}{Dt}\left[r_i\mathbf{e}_i\right] \\ &= \frac{Dy_i}{Dt}\mathbf{e}_i + r_i\frac{D\mathbf{e}_i}{Dt}, \end{aligned} \tag{A.4}$$

where the rate of change of the basis vectors is related to the particle angular velocity $\mathbf{\Omega}$ by $D\mathbf{e}_i/Dt = \mathbf{\Omega} \times \mathbf{e}_i$. As the first term of A.4 is interpreted as the change in $\mathbf{r}$ measured in the particle frame with respect to $\mathbf{e}_i$, it is simply the particle velocity $\mathbf{w}$. Thus,

$$\frac{D\mathbf{r}}{Dt} = \mathbf{w} + \mathbf{\Omega} \times \mathbf{r}. \tag{A.5}$$

We can now relate the fluid velocity in the particle frame $\mathbf{u}$ to the fluid velocity in the laboratory frame by

$$\begin{aligned} \mathbf{U} &= \mathbf{u} + \frac{D\mathbf{r}}{Dt} \\ \mathbf{U} &= \mathbf{u} + \mathbf{w} + \mathbf{\Omega} \times \mathbf{r}. \end{aligned} \tag{A.6}$$

Applying the same logic as that which led to (A.4), we find

$$\frac{D\mathbf{U}}{Dt} = \frac{DU_i}{Dt}\mathbf{e}_i + U_i\frac{D\mathbf{e}_i}{Dt} \tag{A.7}$$

$$= \left(\frac{D\mathbf{u}}{Dt} + \dot{\mathbf{w}} + \dot{\mathbf{\Omega}} \times \mathbf{r}\right) + \left(\mathbf{\Omega} \times \mathbf{u} + \mathbf{\Omega} \times \mathbf{w} + \mathbf{\Omega} \times \mathbf{\Omega} \times \mathbf{r}\right). \tag{A.8}$$

Recognizing that for a particle of radius a, $r = |\mathbf{r}| \leq a$ fallis inside the particle and, due to the no slip condition $\mathbf{u} = \mathbf{w}$ at the particle surface, we end up with

$$\frac{D\mathbf{U}}{Dt} = \frac{D\mathbf{u}}{Dt} + 2\mathbf{\Omega} \times \mathbf{u} + \dot{\mathbf{w}} + \dot{\mathbf{\Omega}} \times \mathbf{r} + \mathbf{\Omega} \times \mathbf{\Omega} \times \mathbf{r}. \tag{A.9}$$

Returning to (A.2), we can write the Navier-Stokes equations in the particle frame keeping in mind that $\mathbf{w}$ and $\mathbf{\Omega}$ are functions of time only:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} + 2\mathbf{\Omega} \times \mathbf{u} = -\frac{1}{\rho_f}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{g} - \left[\dot{\mathbf{w}} + \dot{\mathbf{\Omega}} \times \mathbf{r} + \mathbf{\Omega} \times \mathbf{\Omega} \times \mathbf{r}\right] \tag{A.10a}$$

$$\nabla \cdot \mathbf{u} = 0. \tag{A.10b}$$

Choosing to insert the change of variables (noting that $\tilde{\mathbf{u}} = \mathbf{u} = 0$ at $r = a$)

$$\mathbf{u} = \tilde{\mathbf{u}} + \frac{1}{10\nu}\left(r^2 - a^2\right)\dot{\mathbf{\Omega}} \times \mathbf{r} \tag{A.11a}$$

$$p = \tilde{p} + \frac{1}{2}\rho_f \left(\mathbf{\Omega} \times \mathbf{r}\right)^2 + \rho_f \left(\mathbf{g} - \dot{\mathbf{w}}\right) \cdot \mathbf{r} \tag{A.11b}$$

into the right-hand side of (A.10), we find

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} + 2\mathbf{\Omega} \times \mathbf{u} = -\frac{1}{\rho_f}\nabla \tilde{p} + \nu \nabla^2 \tilde{\mathbf{u}} \tag{A.12a}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{A.12b}$$

because of the following:

$$\begin{aligned}
\nu\nabla^2\left[\frac{1}{10\nu}\left(r^2-a^2\right)\dot{\mathbf{\Omega}}\times\mathbf{r}\right] &= \frac{1}{10}\partial_l\partial_l\left[\left(r^2-a^2\right)\varepsilon_{ijk}\dot{\Omega}_i r_j\right] \\
&= \frac{1}{10}\partial_l\left[\partial_l r^2\varepsilon_{ijk}\dot{\Omega}_i r_j + \left(r^2-a^2\right)\varepsilon_{ilk}\dot{\Omega}_i\right] \\
\partial_l r^2 = \partial_l\left(r_m r_m\right) &= \partial_l r_m r_m + r_m\partial_l r_m = 2r_l \\
\nu\nabla^2\left[\frac{1}{10\nu}\left(r^2-a^2\right)\dot{\mathbf{\Omega}}\times\mathbf{r}\right] &= \frac{1}{10}\partial_l\left[2r_l\varepsilon_{ijk}\dot{\Omega}_i r_j + \left(r^2-a^2\right)\varepsilon_{ilk}\dot{\Omega}_i\right] \\
&= \frac{1}{10}\left[2\delta_{ll}\varepsilon_{ijk}\dot{\Omega}_i r_j + 2\varepsilon_{ilk}\dot{\Omega}_i r_l + 2r_l\varepsilon_{ilk}\dot{\Omega}_i\right] \\
&= \varepsilon_{ijk}\dot{\Omega}_i r_j \\
\nu\nabla^2\left[\frac{1}{10\nu}\left(r^2-a^2\right)\dot{\mathbf{\Omega}}\times\mathbf{r}\right] &= \dot{\mathbf{\Omega}}\times\mathbf{r}
\end{aligned} \tag{A.13a}$$

$$\begin{aligned}
\nabla\cdot\left[\frac{1}{10\nu}\left(r^2-a^2\right)\dot{\mathbf{\Omega}}\times\mathbf{r}\right] &= \frac{1}{10\nu}\partial_k\left[\left(r^2-a^2\right)\varepsilon_{ijk}\dot{\Omega}_i r_j\right] \\
&= \frac{1}{10\nu}\left[\partial_k r^2\varepsilon_{ijk}\dot{\Omega}_i r_j + \left(r^2-a^2\right)\varepsilon_{ijk}\partial_k\dot{\Omega}_i r_j + \left(r^2-a^2\right)\varepsilon_{ikk}\dot{\Omega}\right] \\
&= \frac{1}{10\nu}2r_k\varepsilon_{ijk}\dot{\Omega}_i r_j \\
&= \frac{1}{5\nu}\varepsilon_{ijk}\dot{\Omega}_i r_j r_k \\
\nabla\cdot\left[\frac{1}{10\nu}\left(r^2-a^2\right)\dot{\mathbf{\Omega}}\times\mathbf{r}\right] &= 0
\end{aligned} \tag{A.13b}$$

$$
\begin{aligned}
-\frac{1}{\rho_f}\nabla\left[\frac{1}{2}\rho_f\left(\boldsymbol{\Omega}\times\mathbf{r}\right)^2\right] &= \frac{1}{2}\partial_p\left(\varepsilon_{ijk}\Omega_i r_j\ \varepsilon_{lmk}\Omega_l r_m\right)\\
&= -\frac{1}{2}\partial_p\left[\Omega_i r_j \Omega_l r_m\left(\delta_{il}\delta_{jm}-\delta_{im}\delta_{jl}\right)\right]\\
&= -\frac{1}{2}\partial_p\left(\Omega_i\Omega_i r_j r_j - \Omega_i\Omega_j r_i r_j\right)\\
&= -\Omega_i\Omega_i r_j - \Omega_i\Omega_j r_i\\
&= -\left(\delta_{il}\delta_{jm}-\delta_{im}\delta_{jl}\right)\Omega_i\Omega_l r_m\\
&= -\varepsilon_{ijk}\varepsilon_{lmk}\Omega_i\Omega_l r_m\\
&= \varepsilon_{ikj}\Omega_i\ \varepsilon_{lmk}\Omega_l r_m\\
-\frac{1}{\rho_f}\nabla\left[\frac{1}{2}\rho_f\left(\boldsymbol{\Omega}\times\mathbf{r}\right)^2\right] &= \boldsymbol{\Omega}\times\boldsymbol{\Omega}\times\mathbf{r} \qquad \text{(A.13c)}
\end{aligned}
$$

$$
\begin{aligned}
-\frac{1}{\rho_f}\nabla\left[\rho_f\left(\mathbf{g}-\dot{\mathbf{w}}\right)\cdot\mathbf{r}\right] &= -\partial_j\left(g_i-\dot{w}_i\right)r_j\\
&= \dot{w}_i - g_i\\
-\frac{1}{\rho_f}\nabla\left[\rho_f\left(\mathbf{g}-\dot{\mathbf{w}}\right)\cdot\mathbf{r}\right] &= \dot{\mathbf{w}}-\mathbf{g}. \qquad \text{(A.13d)}
\end{aligned}
$$

Now, considering (A.10), we argue that $\mathbf{u}=0$ at the surface of the particle due to the no-slip condition. Further, due to continuity, there is a region of fluid near the surface in which the left-hand side of (A.10) is small. Therefore, in a region near $r=a$, we may neglect the left-hand side of (A.10) to approximate $\tilde{\mathbf{u}}$ and $\tilde{p}$ as

$$\nabla\cdot\tilde{\mathbf{u}}=0, \qquad (2.12)$$

$$0=-\nabla\tilde{p}+\mu\nabla^2\tilde{\mathbf{u}} \qquad (2.13)$$

which are the Stokes equations. It is worth reiterating that the region in which (2.12) and (2.13) are a good approximation to (A.10) decreases in size as particle Reynolds number $\mathrm{Re}_p = 2p\tilde{u}_0/\nu$ increases (with $\tilde{u}_0$ some relevant reference velocity).

Appendix B

Spherical vector harmonics

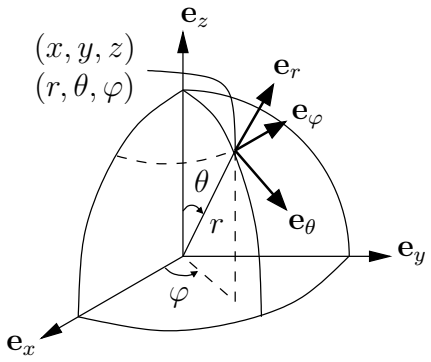


Figure B.1: Spherical coordinate system convention.

Here we record in more detail the implementation of spherical vector harmonics, in which we use the spherical coordinate system convention illustrated in Figure B.1. Recall the following transformations between Cartesian and spherical coordinate

systems:

$$r = \sqrt{x^2 + y^2 + z^2}, \qquad \theta = \arccos\left(\frac{z}{r}\right), \qquad \varphi = \arccos\left(\frac{x}{\sqrt{x^2 + y^2}}\right) \tag{B.1a}$$

$$x = r\sin\theta\cos\phi, \qquad y = r\sin\theta\sin\phi, \qquad z = r\cos\theta. \tag{B.1b}$$

The spherical vector harmonics take the form

$$\begin{aligned} p_l &= \left(\frac{r}{a}\right)^l \sum_{m=-l}^{l} p_{lm} Y_l^m(\theta, \varphi) \\ &= \left(\frac{r}{a}\right)^l \left[p_{l0}^{\Re} N_l^0 P_l^0 + 2\sum_{m=1}^{l} N_l^m P_l^m \left(p_{lm}^{\Re} \cos m\varphi - p_{lm}^{\Im} \sin m\varphi \right) \right], \end{aligned} \tag{2.16}$$

where

$$Y_l^m(\theta, \varphi) = N_l^m P_l^m e^{im\varphi}; \qquad -l \le m \le l, \tag{2.17}$$

with

$$N_l^m = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}}, \tag{2.18}$$

and associated Legendre functions P_l^m is a surface harmonic of order l. Refer to Prosperetti (2011) for extensive discussion on the topic. We list the first five P_l^m in Table B.1 taking the convention that includes the $(-1)^m$ factor when writing P_l^m in terms of the unassociated Legendre functions, namely,

$$P_l^m(\cos\theta) = (-1)^m \sin^{m/2}\theta \frac{d^m}{d(\cos\theta)^m} P_l(\cos\theta). \tag{B.2}$$

It is evident from (2.15) that we will also need the gradient of a solid harmonic, namely,

$$\nabla p_l = \frac{l}{r} p_l \mathbf{e}_r + \left(\frac{r}{a}\right)^{l-1} \frac{1}{a} \sum_{m=-l}^{l} p_{nm} \partial_\theta Y_l^m \mathbf{e}_\theta + \left(\frac{r}{a}\right)^{l-1} \frac{1}{a\sin\theta} \sum_{m=-l}^{l} p_{nm} \partial_\varphi Y_l^m \mathbf{e}_\varphi, \tag{B.3}$$

l	m	$P_l^m(\cos\theta)$
0	0	1
1	0	$\cos\theta$
1	1	$-\sin\theta$
2	0	$\frac{1}{2}(3\cos^2\theta - 1)$
2	1	$-3\cos\theta\sin\theta$
2	2	$3\sin^2\theta$
3	0	$\frac{1}{2}\cos\theta(5\cos^2\theta - 3)$
3	1	$-\frac{3}{2}\sin\theta(5\cos^2\theta - 1)$
3	2	$15\cos\theta\sin^2\theta$
3	3	$-15\sin^3\theta$
4	0	$\frac{1}{8}(35\cos^4\theta - 30\cos^2\theta + 3)$
4	1	$-\frac{5}{2}\cos\theta\sin\theta(7\cos^2\theta - 3)$
4	2	$\frac{15}{2}\sin^2\theta(7\cos^2\theta - 1)$
4	3	$-105\cos\theta\sin^3\theta$
4	4	$105\sin^4\theta$
5	0	$\frac{1}{8}\cos\theta(63\cos^4\theta - 70\cos^2\theta + 15)$
5	1	$-\frac{15}{8}\sin\theta(21\cos^4\theta - 14\cos^2\theta + 1)$
5	2	$\frac{105}{2}\cos\theta\sin^2\theta(3\cos^2\theta - 1)$
5	3	$-\frac{105}{2}\sin^3\theta(9\cos^2\theta - 1)$
5	4	$945\cos\theta\sin^4\theta$
5	5	$-945\sin^5\theta$

Table B.1: Associated Legendre functions up to $l = 5$.

from which

$$(\nabla p_l)\cdot\mathbf{e}_r = \frac{l}{a}\left(\frac{r}{a}\right)^{l-1}\left[p_{n0}^{\Re}N_l^0P_l^0 + 2\sum_{m=1}^{l}N_l^mP_l^m\left(p_{nm}^{\Re}\cos m\varphi - p_{nm}^{\Im}\sin m\varphi\right)\right] \quad \text{(B.4a)}$$

$$\begin{aligned}(\nabla p_l)\cdot\mathbf{e}_\theta =& \frac{1}{a}\left(\frac{r}{a}\right)^{l-1}\left[p_{n0}^{\Re}N_l^0\partial_\theta P_l^0\right.\\ &\left.+2\sum_{m=1}^{l}N_l^m\partial_\theta P_l^m\left(p_{nm}^{\Re}\cos m\varphi - p_{nm}^{\Im}\sin m\varphi\right)\right]\end{aligned} \quad \text{(B.4b)}$$

$$(\nabla p_l)\cdot\mathbf{e}_\varphi = \frac{1}{a\sin\theta}\left(\frac{r}{a}\right)^{l-1}\left[-2\sum_{m=1}^{l}mN_l^m\partial_\theta P_l^m\left(p_{nm}^{\Re}\sin m\varphi + p_{nm}^{\Im}\cos m\varphi\right)\right]. \quad \text{(B.4c)}$$

We will also need the curl of $(\mathbf{r}p_l)$, namely,

$$\nabla \times (\mathbf{r}p_l) = \frac{1}{\sin\theta}\partial_\varphi p_l \mathbf{e}_\theta - \partial_\theta p_l \mathbf{e}_\varphi, \tag{B.5}$$

from which

$$[\nabla \times (\mathbf{r}p_l)] \cdot \mathbf{e}_r = 0 \tag{B.6a}$$

$$[\nabla \times (\mathbf{r}p_l)] \cdot \mathbf{e}_\theta = \frac{1}{\sin\theta}\left(\frac{r}{a}\right)^l \left[-2\sum_{m=1}^{l} m N_l^m P_l^m \left(p_{nm}^{\Re}\sin l\varphi + p_{nm}^{\Im}\cos m\varphi\right)\right] \tag{B.6b}$$

$$\begin{aligned}[\nabla \times (\mathbf{r}p_l)] \cdot \mathbf{e}_\varphi = -\left(\frac{r}{a}\right)^l \Bigg[& p_{n0}^{\Re} N_l^0 \partial_\theta P_l^0 \\ & +2\sum_{m=1}^{l} m N_l^m \partial_\theta P_l^m \left(p_{nm}^{\Re}\cos l\varphi - p_{nm}^{\Im}\sin m\varphi\right)\Bigg]\end{aligned} \tag{B.6c}$$

where

$$\partial_\theta P_l^m = \frac{l-m+1}{\sin\theta}P_{l+1}^m - (l+1)\frac{\cos\theta}{\sin\theta}P_l^m. \tag{B.7}$$

Putting all of this together and noticing a pattern, we can simplify our notation by defining the following functions of p_l, ϕ_l, and χ_l:

$$\mathcal{X}(p_n) = p_{n0}^{\Re} N_n^0 P_n^0 + 2\sum_{m=1}^{n} N_n^m P_n^m \left(p_{nm}^{\Re}\cos m\varphi - p_{nm}^{\Im}\sin m\varphi\right) \tag{B.8a}$$

$$\mathcal{Y}(p_n) = p_{n0}^{\Re} N_n^0 \partial_\theta P_n^0 + 2\sum_{m=1}^{n} N_n^m \partial_\theta P_n^m \left(p_{nm}^{\Re}\cos m\varphi - p_{nm}^{\Im}\sin m\varphi\right) \tag{B.8b}$$

$$\mathcal{Z}(p_n) = -\frac{2}{\sin\theta}\sum_{m=1}^{n} m N_n^m P_n^m \left(p_{nm}^{\Re}\sin m\varphi + p_{nm}^{\Im}\cos m\varphi\right). \tag{B.8c}$$

This leaves us with the following simplified equations for the components of velocity and pressure (2.15):

$$\begin{aligned}\tilde{u}_r(r,\theta,\phi) = &\frac{1}{2}\frac{\nu}{a}\sum_{n=0}^{\infty}\frac{n}{2n+3}\left(\frac{r}{a}\right)^{n+1}\mathcal{X}(p_n)\\ &+\frac{\nu}{a}\sum_{n=1}^{\infty}\left(\frac{r}{a}\right)^{n-1}\mathcal{X}(\varphi_n)\\ &+\frac{\nu}{4a}\sum_{n=1}^{\infty}n\left[\frac{2n+1}{2n+3}\left(\frac{a}{r}\right)^2-1\right]\left(\frac{a}{r}\right)^n\mathcal{X}(p_n)\\ &+\frac{\nu}{2a}\sum_{n=1}^{\infty}n\left[2n-1-(2n+1)\left(\frac{a}{r}\right)^2\right]\left(\frac{a}{r}\right)^{n+2}\mathcal{X}(\phi_n)\end{aligned} \tag{B.9a}$$

$$\begin{aligned}\tilde{u}_\theta(r,\theta,\phi) = &\frac{\nu}{2a}\sum_{n=0}^{\infty}\frac{n+3}{(n+1)(2n+3)}\left(\frac{r}{a}\right)^{n+1}\mathcal{Y}(p_n)\\ &+\frac{\nu}{a}\sum_{n=1}^{\infty}\left(\frac{r}{a}\right)^{n-1}\left[\mathcal{Y}(\phi_n)+\mathcal{Z}(\chi_n)\right]\\ &+\frac{\nu}{4a}\sum_{n=1}^{\infty}\frac{1}{n+1}\left[n-2-\frac{n(2n+1)}{2n+3}\left(\frac{a}{r}\right)^2\right]\left(\frac{a}{r}\right)^n\mathcal{Y}(p_n)\\ &+\frac{\nu}{2a}\sum_{n=1}^{\infty}\frac{1}{n+1}\left[(n-2)(2n+1)\left(\frac{r}{a}\right)^2-n(2n-1)\right]\left(\frac{a}{r}\right)^{n+2}\mathcal{Y}(\phi_n)\\ &-\frac{\nu}{a}\sum_{n=1}^{\infty}\left(\frac{a}{r}\right)^{n+1}\mathcal{Z}(\chi_n)\end{aligned} \tag{B.9b}$$

$$\begin{aligned}\tilde{u}_\phi(r,\theta,\phi) = &\frac{\nu}{2a}\sum_{n=0}^{\infty}\frac{n+3}{(n+1)(2n+3)}\left(\frac{r}{a}\right)^{n+1}\mathcal{Z}(p_n)\\ &+\frac{\nu}{a}\sum_{n=1}^{\infty}\left(\frac{r}{a}\right)^{n-1}\left[\mathcal{Z}(\phi_n)-\mathcal{Y}(\chi_n)\right]\\ &+\frac{\nu}{4a}\sum_{n=1}^{\infty}\frac{1}{n+1}\left[n-2-\frac{n(2n+1)}{2n+3}\left(\frac{a}{r}\right)^2\right]\left(\frac{a}{r}\right)^n\mathcal{Z}(p_n)\\ &+\frac{\nu}{2a}\sum_{n=1}^{\infty}\frac{1}{n+1}\left[(n-2)(2n+1)\left(\frac{r}{a}\right)^2-n(2n-1)\right]\left(\frac{a}{r}\right)^{n+2}\mathcal{Z}(\phi_n)\\ &+\frac{\nu}{a}\sum_{n=1}^{\infty}\left(\frac{a}{r}\right)^{n+1}\mathcal{Y}(\chi_n)\end{aligned} \tag{B.9c}$$

$$\begin{aligned}\tilde{p}(r,\theta,\phi) = &\sum_{n=0}^{\infty}\left[1-\frac{n(2n-1)}{2(n+1)}\left(\frac{a}{r}\right)^{2n+1}\right]\left(\frac{r}{a}\right)^n\mathcal{X}(p_n)\\ &-\frac{n(2n-1)(2n+1)}{n+1}\left(\frac{a}{r}\right)^n+\mathcal{X}(\phi_n).\end{aligned} \tag{B.9d}$$

Bibliography

Apte, S.V., Martin, M., Patankar, N.A., 2009. A numerical method for fully resolved simulation (FRS) of rigid particle-flow interactions in complex flows. J. Comput. Phys. 228, 2712–2738. doi:10.1016/j.jcp.2008.11.034.

Balachandar, S., Eaton, J.K., 2010. Turbulent dispersed multiphase flow. Ann. Rev. Fluid Mech. 42, 111–133. doi:10.1146/annurev.fluid.010908.165243.

Barnocky, G., Davis, R.H., 1988. Elastohydrodynamic collision and rebound of spheres: Experimental verification. Phys. Fluids 31, 1324–1329. doi:10.1063/1.866725.

Batchelor, G.K., 2000. An Introduction to Fluid Dynamics. Cambridge University Press, New York, NY.

Botto, L., Prosperetti, A., 2012. A fully resolved numerical simulation of turbulent flow past one or several spherical particles. Phys. Fluids 24. doi:10.1063/1.3678336.

Brady, J.F., Bossis, G., 1988. Stokesian dynamics. Ann. Rev. Fluid Mech. 20, 111–157. doi:10.1146/annurev.fl.20.010188.000551.

Brown, D.L., Cortez, R., Minion, M.L., 2001. Accurate projection methods for the incompressible Navier-Stokes equations. J. Comput. Phys. 168, 464–499. doi:10.1006/jcph.2001.6715.

van Buijtenen, M.S., van Dijk, W.J., Deen, N.G., Kuipers, J., Leadbeater, T., Parker, D.J., 2011. Numerical and experimental study on multiple-spout fluidized beds. Chem. Eng. Sci. 66, 2368–2376. doi:10.1016/j.ces.2011.02.055.

Carroll, P.L., Blanquart, G., 2013. A proposed modification to lundgren's physical space velocity forcing method for isotropic turbulence. Phys. Fluids 25. doi:10.1063/1.4826315.

ten Cate, A., Nieuwstad, C.H., Derksen, J.J., den Akker, H.E.A.V., 2002. Particle imaging velocimetry experiments and lattice-boltzmann simulations on a single sphere settling under gravity. Phys. Fluids 14, 4012–4025. doi:10.1063/1.1512918.

Chorin, A.J., 1967. The numerical solution of the Navier-Stokes equations for an incompressible fluid. Bull. Am. Math. Soc. 73, 928–931. doi:10.1090/S0002-9904-1967-11853-6.

Crowe, C.T., Schwarzkopf, J.D., Sommerfeld, M., Tsuji, Y., 2012. Multiphase Flows with Droplets and Particles. 2nd ed., CRC Press, Boca Raton, USA.

Dalton, S., Bell, N., Olson, L., Garland, M., 2014. Cusp: Generic parallel algorithms for sparse matrix and graph computations. URL: `http://cusplibrary.github.io/`. version 0.5.0.

Davis, R.H., Rager, D.A., Good, B.T., 2002. Elastohydrodynamic rebound of spheres from coated surfaces. J. of Fluid Mech. 468, 107–119. doi:`10.1017/S0022112002001489`.

Davis, R.H., Serayssol, J.M., Hinch, E.J., 1986. The elastohydrodynamic collision of two spheres. J. of Fluid Mech. 163, 479–497. doi:`10.1017/S0022112086002392`.

Deen, N.G., Annaland, M.V.S., der Hoef, M.A.V., Kuipers, J.A.M., 2007. Review of discrete particle modeling of fluidized beds. Chem. Eng. Sci. 62, 28–44. doi:`10.1016/j.ces.2006.08.014`. fluidized Bed Applications.

Doostmohammadi, A., Ardekani, A.M., 2015. Suspension of solid particles in a density stratified fluid. Phys. Fluids 27. doi:`10.1063/1.4907875`.

Elghobashi, S., Truesdell, G.C., 1992. Direct simulation of particle dispersion in a decaying isotropic turbulence. J. Fluid Mech. 242, 655–700. doi:`10.1017/S0022112092002532`.

Ferziger, J.H., Perić, M., 2002. Computational Methods for Fluid Dynamics. 3rd ed., Springer, Berlin.

Fortin, P., Athanassoula, E., Lambert, J.C., 2011. Comparisons of different codes

for galactic n-body simulations. Astron. Astrophys. 531, A120. doi:10.1051/0004-6361/201015933.

Fox, R.O., 2012. Large-eddy-simulation tools for multiphase flows. Ann. Rev. Fluid Mech. 44, 47–76. doi:10.1146/annurev-fluid-120710-101118.

Franka, N.P., Heindel, T.J., Battaglia, F., 2007. Visualizing cold-flow fluidized beds with X-rays, in: ASME Int. Mech. Eng. Congr. Proc., pp. 99–105. doi:10.1115/IMECE2007-43073.

Gondret, P., Lance, M., Petit, L., 2002. Bouncing motion of spherical particles in fluids. Phys. Fluids 14, 643–652. doi:dx.doi.org/10.1063/1.1427920.

Goudie, A.S., 2008. The history and nature of wind erosion in deserts. Ann. Rev. Earth Pl. Sci. 36, 97–119. doi:10.1146/annurev.earth.36.031207.124353.

Govindaraju, N., Lloyd, B., Dotsenko, Y., Smith, B., Manferdelli, J., 2008. High performance discrete fourier transforms on graphics processors, in: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, IEEE. pp. 1–12.

Grabowski, W.W., Wang, L.P., 2013. Growth of cloud droplets in a turbulent environment. Annu. Rev. Fluid Mech. 45, 293–324. doi:10.1146/annurev-fluid-011212-140750.

Grace, J.R., Leckner, B., Zhu, J., Cheng, Y., 2006. Fluidized beds, in: Crowe, C.T. (Ed.), Multiphase Flow Handbook. CRC Press, Boca Raton, FL, pp. 5–1–5–93.

Gudmundsson, K., Prosperetti, A., 2013. Improved procedure for the computation of Lamb's coefficients in the physalis method for particle simulation. J. Comput. Phys. 234, 44–59. doi:10.1016/j.jcp.2012.08.049.

Guha, A., 2008. Transport and deposition of particles in turbulent and laminar flow. Ann. Rev. Fluid Mech. 40, 311–341. doi:10.1146/annurev.fluid.40.111406.102220.

van der Hoef, M.A., van Sint Annaland, M., Deen, N.G., Kuipers, J., 2008. Numerical simulation of dense gas-solid fluidized beds: A multiscale modeling strategy. Ann. Rev. Fluid Mech. 40, 47–70. doi:10.1146/annurev.fluid.40.111406.102130.

Jackson, R., 2000. The Dynamics of Fluidized Particles. Cambridge University Press, Cambridge.

Jeffrey, D.J., 1982. Low-reynolds-number flow between converging spheres. Mathematika 29, 58–66. doi:10.1112/S002557930001216X.

Ji, C., Munjiza, A., Avital, E., Xu, D., Williams, J., 2014. Saltation of particles in turbulent channel flow. Phys. Rev. E 89, 052202. doi:10.1103/PhysRevE.89.052202.

Jorgensen, S.E., Johnsen, I., 1981. Principles of Environmental Science and Technology. Elsevier Scientific Publishing Co., New York, NY.

Joseph, G.G., Hunt, M.L., 2004. Oblique particle-wall collisions in a liquid. J. Fluid Mech. 510, 71–93. doi:10.1017/S002211200400919X.

Joseph, G.G., Zenit, R., Hunt, M.L., Rosenwinkel, A.M., 2001. Particle-wall collisions in a viscous fluid. J. Fluid Mech. 433, 329–346. doi:10.1017/S0022112001003470.

Joshi, J.B., Nandakumar, K., 2015. Computational modeling of multiphase reactors. Ann. Rev. Chem. Biom. Eng. 6, 347–378. doi:10.1146/annurev-chembioeng-061114-123229.

Katz, J., Sheng, J., 2010. Applications of holography in fluid mechanics and particle dynamics. Ann. Rev. Fluid Mech. 42, 531–555. doi:10.1146/annurev-fluid-121108-145508.

Kempe, T., Fröhlich, J., 2012a. An improved immersed boundary method with direct forcing for the simulation of particle laden flows. J. Comput. Phys. 231, 3663–3684. doi:10.1016/j.jcp.2012.01.021.

Kempe, T., Fröhlich, J., 2012b. Collision modelling for the interface-resolved simulation of spherical particles in viscous fluids. J. Fluid Mech. 709, 445–489. doi:10.1017/jfm.2012.343.

Kidanemariam, A.G., Uhlmann, M., 2014. Direct numerical simulation of pattern formation in subaqueous sediment. J. Fluid Mech. 750. doi:10.1017/jfm.2014.284.

Kim, J., Moin, P., 1985. Application of a fractional-step method to incompressible Navier-Stokes equations. J. Comput. Phys. 59, 308–323. doi:10.1016/0021-9991(85)90148-2.

Kim, S., Karilla, S.J., 1991. Microhydrodynamics: Principles and Selected Applications. Sect. 4.2, Butterworth-Heinemann, Boston, MA.

Koch, D.L., Hill, R.J., 2001. Inertial effects in suspension and porous-media flows. Ann. Rev. Fluid Mech. 33, 619–647. doi:10.1146/annurev.fluid.33.1.619.

Kriegeskorte, N., 2015. Deep neural networks: A new framework for modeling biological vision and brain information processing. Annual Review of Vision Science 1, 417–446. doi:10.1146/annurev-vision-082114-035447.

Laín, S., Sommerfeld, M., Kussin, J., 2002. Experimental studies and modelling of four-way coupling in particle-laden horizontal channel flow. Int. J. Heat Fluid Fl. 23, 647–656. doi:10.1016/S0142-727X(02)00160-1.

Lamb, H., 1932. Hydrodynamics. Art. 336. 6th ed., Dover Publications, New York, NY.

Leal, L.G., 1980. Particle motions in a viscous fluid. Ann. Rev. Fluid Mech. 12, 435–476. doi:10.1146/annurev.fl.12.010180.002251.

Lebedev, V.I., 1976. Quadratures on a sphere. USSR Comput. Math. Math. Phys. 16, 10–24. doi:10.1016/0041-5553(76)90100-2.

Li, X., Hunt, M.L., Colonius, T., 2012. A contact model for normal immersed collisions between a particle and a wall. J. Fluid Mech. 691, 123–145. doi:10.1017/jfm.2011.461.

Liu, Q., Prosperetti, A., 2010. Wall effects on a rotating sphere. J. Fluid Mech. 657, 1–21. doi:10.1017/S002211201000128X.

Liu, Q., Prosperetti, A., 2011. Pressure-driven flow in a channel with porous walls. J. Fluid Mech. 679, 77–100. doi:10.1017/jfm.2011.124.

Lucci, F., Ferrante, A., Elghobashi, S., 2010. Modulation of isotropic turbulence by particles of taylor length-scale size. J. Fluid Mech. 650, 5–55. doi:10.1017/S0022112009994022.

Maxey, M.R., 1987. The gravitational settling of aerosol particles in homogeneous turbulence and random flow fields. J. Fluid Mech. 174, 441–65. doi:10.1017/S0022112087000193.

McLaughlin, M., 1968. An experimental study of particle-wall collision relating to flow of solid particles in a fluid. Master's thesis. California Institute of Technology. Pasadena, California.

Mehrabadi, M., Tenneti, S., Garg, R., Subramaniam, S., 2015. Pseudo-turbulent gas-phase velocity fluctuations in homogeneous gassolid flow: fixed particle assemblies

and freely evolving suspensions. J. Fluid Mech. 770, 210–246. doi:10.1017/jfm.2015.146.

Mittal, R., Iaccarino, G., 2005. Immersed boundary methods. Ann. Rev. Fluid Mech. 37, 239–261. doi:10.1146/annurev.fluid.37.061903.175743.

Mordant, N., Pinton, J.F., 2000. Velocity measurement of a settling sphere. Eur. Phys. J. B 18, 343–352. doi:10.1007/PL00011074.

Naso, A., Prosperetti, A., 2010. The interaction between a solid particle and a turbulent flow. New J. Phys. 12, 033040. doi:10.1088/1367-2630/12/3/033040.

Peskin, C.S., 1977. Numerical analysis of blood flow in the heart. J. Comput. Phys. 25, 220–252. doi:10.1016/0021-9991(77)90100-0.

Picano, F., Breugem, W.P., Brandt, L., 2015. Turbulent channel flow of dense suspensions of neutrally buoyant spheres. J. Fluid Mech. 764, 463–487. doi:10.1017/jfm.2014.704.

Prosperetti, A., 2011. Advanced Mathematics for Applications. Cambridge University Press, Cambridge, United Kingdom.

Prosperetti, A., 2015. Life and death by boundary conditions. J. Fluid Mech. 768, 1–4. doi:10.1017/jfm.2015.32.

Prosperetti, A., Oğuz, H.N., 2001. Physalis: A new O(N) method for the numerical

simulation of disperse systems: Potential flow of spheres. J. Comput. Phys. 167, 196–216. doi:10.1006/jcph.2000.6667.

Prosperetti, A., Tryggvason, G. (Eds.), 2009. Computational Methods for Multiphase Flow. Cambridge University Press, Cambridge, United Kingdom.

Rautenbach, C., Melaaen, M.C., Halvorsen, B.M., 2011. Investigating the influence of fines in fluidized bed reactors using 3D ECT images, in: WIT Trans. Eng. Sci., Kos, Greece. pp. 141–151. doi:10.2495/MPF110121.

Richardson, J.F., Zaki, W.N., 1954. Sedimentation and fluidisation: Part 1. Trans. Instn. Chem. Engrs. 32, 35–52.

Rosales, C., Meneveau, C., 2005. Linear forcing in numerical simulations of isotropic turbulence: Physical space implementations and convergence properties. Phys. Fluids 17. doi:10.1063/1.2047568.

Saleh, J.M., 2002. Fluid Flow Handbook. McGraw-Hill Education, New York, NY.

Sarkar, S., van der Hoef, M.A., Kuipers, J.A.M., 2009. Fluid-particle interaction from lattice boltzmann simulations for flow through polydisperse random arrays of spheres. Chem. Eng. Sci. 64, 2683–2691. doi:10.1016/j.ces.2009.02.045.

Shaffer, F., Gopalan, B., 2013. The science and beauty of fluidization. APS Gallery of Fluid Motion entry number 102371 .

Sierakowski, A.J., 2016a. Gpu-centric resolved-particle disperse two-phase flow simulation using the Physalis method. Comput. Phys. Commun. In press.

Sierakowski, A.J., 2016b. A nonlinearly damped hertzian contact model for resolved-particle flow simulations. Int. J. Multiphase Flow In preparation.

Sierakowski, A.J., Prosperetti, A., 2016. Resolved-particle simulation by the Physalis method: Enhancements and new capabilities. J. Comput. Phys. 309, 164–184. doi:10.1016/j.jcp.2015.12.057.

Silberstein, M., Schuster, A., Geiger, D., Patney, A., Owens, J.D., 2008. Efficient computation of sum-products on gpus through software-managed cache, in: Proceedings of the 22nd Annual International Conference on Supercomputing, ACM, New York, NY, USA. pp. 309–318. doi:10.1145/1375527.1375572.

Simeonov, J.A., Calantoni, J., 2012. Modeling mechanical contact and lubrication in direct numerical simulations of colliding particles. Int. J. Multiphase Flow 46, 38–53. doi:10.1016/j.ijmultiphaseflow.2012.05.008.

Stickel, J.J., Powell, R.L., 2005. Fluid mechanics and rheology of dense suspensions. Annual Review of Fluid Mechanics 37, 129–149. doi:10.1146/annurev.fluid.36.050802.122132.

Stokes, G.G., 1851. On the effect of the internal friction of fluids on the motion of pendulums. T. Camb. Philos. Soc. 9, 8.

Sundaram, S., Collins, L.R., 1996. Numerical considerations in simulating a turbulent suspension of finite-volume farticles. J. Comp. Phys. 124, 337–350. doi:10.1006/jcph.1996.0064.

Sundaresan, S., 2003. Instabilities in fluidized beds. Ann. Rev. Fluid Mech. 35, 63–88. doi:10.1146/annurev.fluid.35.101101.161151.

Takagi, S., Oğuz, H.N., Zhang, Z., Prosperetti, A., 2003. Physalis: A new method for particle simulation: Part II: Two-dimensional Navier-Stokes flow around cylinders. J. Comp. Phys. 187, 371–390. doi:10.1016/S0021-9991(03)00077-9.

Tenneti, S., Subramaniam, S., 2014. Particle-resolved direct numerical simulation for gas-solid flow model development. Ann. Rev. Fluid Mech. 46, 199–230. doi:10.1146/annurev-fluid-010313-141344.

Tezduyar, T.E., 2001. Finite element methods for flow problems with moving boundaries and interfaces. Arch. Comput. Method. E. 8, 83–130. doi:10.1007/BF02897870.

Tsuji, Y., Tanaka, T., Ishida, T., 1992. Lagrangian numerical simulation of plug flow of cohesionless particles in a horizontal pipe. Powder Technol. 71, 239–250. doi:10.1016/0032-5910(92)88030-L.

Uhlmann, M., 2005. An immersed boundary method with direct forcing for the

simulation of particulate flows. J. Comput. Phys. 209, 448–476. doi:10.1016/j.jcp.2005.03.017.

Uhlmann, M., 2008. Interface-resolved direct numerical simulation of vertical particulate channel flow in the turbulent regime. Phys. Fluids 20. doi:10.1063/1.2912459.

Wilcox, D.C., 1998. Turbulence Modeling for CFD. 2nd ed., DCW Industries, La Cañada, CA.

Willgoose, G., 2005. Mathematical modeling of whole landscape evolution. Annual Review of Earth and Planetary Sciences 33, 443–459. doi:10.1146/annurev.earth.33.092203.122610.

Wylie, J.J., Koch, D.L., Ladd, A.J.C., 2003. Rheology of suspensions with high particle inertia and moderate fluid inertia. J. Fluid Mech. 480, 95–118. doi:10.1017/S0022112002003531.

Yang, F.L., Hunt, M.L., 2006. Dynamics of particle-particle collisions in a viscous liquid. Physics of Fluids 18. doi:10.1063/1.2396925.

Yang, J., Stern, F., 2015. A non-iterative direct forcing immersed boundary method for strongly-coupled fluid-solid interactions. J. Comput. Phys. 295, 779–804. doi:10.1016/j.jcp.2015.04.040.

Yang, W.C., 2003. Handbook of Fluidization and Fluid-Particle Systems. Marcel Dekker, New York, NY.

Zeng, L., Balachandar, S., Najjar, F.M., 2010. Wake response of a stationary finite-sized particle in a turbulent channel flow. Int. J. Multiphase Flow 36, 406–422. doi:10.1016/j.ijmultiphaseflow.2010.01.001.

Zhang, Q., Prosperetti, A., 2009. Pressure-driven flow in a two-dimensional channel with porous walls. J. Fluid Mech. 631, 1–21. doi:10.1017/S0022112009005837.

Zhang, Z., Botto, L., Prosperetti, A., 2006. Microstructural effects in a fully-resolved simulation of 1,024 sedimenting spheres, in: IUTAM Symposium on Computational Approaches to Multiphase Flow, pp. 1–10.

Zhang, Z., Prosperetti, A., 2005. A second-order method for three-dimensional particle simulation. J. Comput. Phys. 210, 292–324. doi:10.1016/j.jcp.2005.04.009.

Curriculum Vitae

Adam J. Sierakowski was born in Baltimore, Maryland on 10 January 1988. He graduated from the Johns Hopkins University in 2010 with a Bachelor of Science degree in Mechanical Engineering with an Aerospace Engineering concentration and a minor in Mathematics. As a National Science Foundation IGERT Fellow (Integrative Graduate Education and Research Traineeship) in the Johns Hopkins University Modeling Complex Systems program, Adam obtained a Master of Science degree in Mechanical Engineering in 2015 and a Doctor of Philosophy degree in Mechanical Engineering in 2016 under his advisor, Professor Andrea Prosperetti.

Adam has extensive experience in developing computational simulation tools for operation on graphics processing units and other high performance computing architectures. In addition to performing the work presented in this dissertation, Adam designed, built, and administrated a multi-node code development cluster for his research group, helped mentor junior Doctoral students, and assisted in the submission of multiple research grant proposals. Dedicated to the advancement of our future scientists and engineers, Adam designed and taught two undergraduate scientific

computing courses in the Mechanical Engineering Department at the Johns Hopkins University and participated in many events encouraging the youth of Baltimore City to pursue studies in Science, Technology, Engineering, and Mathematics.

Adam is husband to Professor Claire VerHulst, Ph.D. of the United States Military Academy and is a four-time USA Triathlon All-American athlete.